VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2022 Pavel Prášil

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

MODERNIZACE VÝUKOVÝCH ÚLOH KURZU LOGICKÉ OBVODY A SYSTÉMY

MODERNIZATION OF EDUCATIONAL EXERCISES OF THE COURSE LOGICAL CIRCUITS AND SYSTEMS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Pavel Prášil

AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Petyovský, Ph.D.

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Pavel Prášil ***ID:*** 216847

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Modernizace výukových úloh kurzu Logické obvody a systémy

POKYNY PRO VYPRACOVÁNÍ:

1. Proveďte rešerši existujících zadání úloh pro kurz Logické obvody a systémy, včetně nově připravovaných úloh se zvukovým generátorem.
2. Zvolte vhodnou existující architekturu asynchronního sériového vysílače / přijímače (UART), případně navrhněte vlastní.
3. Navrhněte vnitřní blokové schéma vysílače i přijímače a implementujte jej do hradlového pole. Proveďte ověření funkčnosti jednotlivých bloků pomocí simulace.
4. Navrhněte blokové uspořádání všech komponent laboratorní úlohy tak, aby bylo možné realizovaný vysílač / přijímač vhodně využít pro demonstraci funkce pomocí programu v PC.
5. Seznamte se s výsledky BP: Realizace výukového vícekanálového zvukového obvodu.
6. Navrhněte a realizujte vhodné řešení, umožňující přehrávání zvukových dat pomocí realizovaného zvukového generátoru a navrženého vysílače / přijímače.
7. Demonstrujte funkčnost řešení a zpracujte podklady pro studentskou laboratorní úlohu realizující asynchronní sériový vysílač / přijímač.
8. Zhodnoťte dosažené výsledky, uveďte výhody a nevýhody řešení a navrhněte další možná rozšíření.

DOPORUČENÁ LITERATURA:

[1] PINKER, J. POUPA, M: Číslicové systémy a jazyk VHDL. 2006, ISBN 80-7300-198-5.

[2] HOMZOVÁ, E.: Realizace výukového vícekanálového zvukového obvodu. Bakalářská práce UAMT FEKT VUT v Brně, 2021.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 23.5.2022

Vedoucí práce: Ing. Petr Petyovský, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zabývá návrhem asynchronního sériového vysílače/přijímače a jeho implementací do hradlového pole. Následně bude návrh využit jako výuková laboratorní úloha kurzu „Logické obvody a systémy“. Práce obsahuje popis vlastností sériového komunikačního rozhraní UART. Součástí práce je výsledný návrh asynchronního sériového vysílače/přijímače a výstupy simulace jednotlivých bloků. Výsledný návrh UART bude využit jako komunikační rozhraní pro přehrávání hudebních dat, pomocí programovatelného zvukového generátoru. Návrh programovatelného vícekanálového zvukového obvodu není součástí této práce, je převzatý z jiné bakalářské práce.

KLÍČOVÁ SLOVA

UART, programovatelný vícekanálový zvukový obvod, FPGA, VHDL

ABSTRACT

This bachelor thesis deals with the design of an asynchronous serial receiver/transmitter and its implementation into the FPGA. The design will be used as a laboratory exercise in the course "Logical circuit and systems". This paper contains a description of the features of serial communication interface UART. The thesis includes the final design of an asynchronous serial receiver/transmitter and the simulation outputs of particular parts. The final design of UART will be used as a communication interface for music playback by the programmable multichannel sound generator. Design of the programmable multichannel sound generator is not a part of this thesis, but it has been taken from another bachelor thesis.

KEYWORDS

UART, programmable multichannel sound generator, FPGA, VHDL

PRÁŠIL, Pavel. *Modernizace výukových úloh kurzu Logické obvody a systémy*. Brno, 2022. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/142706. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 65 s. Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Pavel Prášil
VUT ID autora:	216847
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Modernizace výukových úloh kurzu Logické obvody a systémy

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno 23.5.2022 .....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Petyovskému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat své rodině a přátelům za jejich podporu.

Obsah

Úvod **13**

1 Komunikační sběrnice UART **14**
1.1 Základní popis . . . 14
1.2 Datový rámec UART . . . 14
1.3 Vyrovnávací paměť . . . 15
1.4 Řízení toku dat . . . 15
1.4.1 HW řízení toku dat . . . 15
1.4.2 SW řízení toku dat . . . 16
1.5 Architektura UART mikrokontroléru M68HC11 . . . 16
1.5.1 Funkce Send Break . . . 16
1.5.2 Funkce Queued Idle Character . . . 16
1.5.3 Vzorkování dat přijímačem . . . 17
1.5.4 Detekce start bitu . . . 17

2 Zadání vybraných úloh kurzu BPC-LOS **18**
2.1 Úloha č. 6 - Čítače a rozšířená dělička frekvence . . . 18
2.2 Úloha č. 8 - Konečný stavový automat . . . 18
2.3 Připravovaná úloha se zvukovým generátorem . . . 19

3 Vnitřní blokové schéma navrženého UART a simulace jednotlivých bloků **20**
3.1 Vysílač UART . . . 20
3.1.1 Dělička hodinového signálu . . . 21
3.1.2 Posuvný registr vysílače . . . 24
3.1.3 Řídicí stavový automat vysílače . . . 26
3.2 Přijímač UART . . . 30
3.2.1 Filtr metastabilního děje . . . 30
3.2.2 Detektor start bitu . . . 31
3.2.3 Dělička hodinového signálu . . . 32
3.2.4 Posuvný registr přijímače . . . 33
3.2.5 Řídicí stavový automat přijímače . . . 37
3.2.6 Status registr přijímače . . . 40
3.3 Konfigurace UART . . . 40
3.3.1 Konfigurační registry . . . 40
3.3.2 Nastavení adresy a hodnoty registru . . . 41

4 Realizace rozhraní pro přehrávání zvukových dat pomocí PSG 42
4.1 Blok realizující příjem dat a zpracování kontrolního součtu 42
4.1.1 Formát přijímaných zpráv z PC 43
4.1.2 Formát potvrzovacích zpráv odesílaných vývojovou deskou . . 43
4.2 Blok realizující přepis registrů PSG 43

5 Analýza návrhu pomocí ChipScope 44
5.1 Jednotlivé prvky pro testování pomocí ChipScope 44
5.1.1 ILA core testovací součást 44
5.1.2 ICON core testovací součást 44
5.2 Konfigurace Chipscope 44
5.3 Zobrazení měřených průběhů pomocí ChipScope 46
5.4 Příklady časových průběhů naměřených pomocí ChipScope 46

6 Návrh laboratorní úlohy 48
6.1 Vývojová deska NEXYS3 48
6.2 Převodník FTDI FR232 48
6.3 Laboratorní úloha návrhu programovatelné děličky frekvence 48
6.4 Laboratorní úloha pro spojení s PC 49
6.4.1 Využití periferií vývojové desky k demonstraci funkce UART . 49
6.4.2 Zadání laboratorní úlohy 50

Závěr 51

Literatura 53

Seznam symbolů a zkratek 56

Seznam příloh 57

A Zadání studentské laboratorní úlohy pro návrh programovatelné děličky frekvence UART 58

B Zadání studentské laboratorní úlohy pro dokončení a zprovoznění UART 61

C Obsah elektronické přílohy 64

Seznam obrázků

1.1 Datový rámec UART [15] 14
1.2 Detekce start bitu přijímačem M68HC11 – ideální případ [6] 17
2.1 Lineární zpětnovazební čítač použitý v generátoru šumu [3] 19
3.1 Blokové schéma vysílače 20
3.2 Simulace děličky frekvence hodinového signálu vysílače 24
3.3 Simulace bloku posuvného registru vysílače s odesláním dvou datových rámců a vygenerovaným signálem *reset*. V simulaci je nastavena sudá parita. 26
3.4 Simulace bloku posuvného registru vysílače s odesláním pozastavovacího znaku XOFF a obnovovacího znaku XON, softwarového řízení datového toku 26
3.5 Stavový diagram řídicího stavového automatu vysílače 28
3.6 Simulace bloku řídicího stavového automatu vysílače s odesíláním datového rámce s paritním bitem, bez něj a provedením resetu 29
3.7 Simulace bloku řídicího stavového automatu vysílače s odesláním znaků softwarového řízení toku dat 29
3.8 Blokové schéma přijímače 30
3.9 Simulace dvou metastabilních dějů na vstupu bloku Filter, použitého v přijímači 31
3.10 Simulace funkce bloku Start detector 31
3.11 Simulace děličky frekvence přijímače, s vyznačeným okamžikem spuštění 32
3.12 Detail výstupu simulace děličky frekvence přijímače se zobrazením počítání čítačů generujících výstupní signál *tc_16*. 33
3.13 Ilustrace FIFO bufferu přijímače o zvolené hloubce 8 bajtů 35
3.14 Simulace posuvného registru přijímače s vygenerovaným zarušením dvou příchozích bitů a resetem 36
3.15 Simulace detailního pohledu na vzorkování zarušeného bitu posuvným registrem přijímače 36
3.16 Simulace příjmu dvou datových rámců s paritním bitem, korektním i nekorektním 36
3.17 Simulace funkce osmibajtového FIFO bufferu posuvného registru přijímače 37
3.18 Stavový diagram řídicího stavového automatu přijímače 38
3.19 Simulace bloku řídicího stavového automatu přijímače, s vyznačeným okamžikem pulsu výstupu *ld_buf* a resetu 39

3.20 Simulace bloku řídicího stavového automatu přijímače s vygenerovanou chybnou hodnotou start bitu a stop bitu. Také s přepnutím příjmu paritního bitu signálem *use_par*. 39
4.1 Blokové schéma navrženého rozhraní mezi UART a PSG (Přivedení synchronizačního signálu *clk* a resetu je pro některé komponenty zanedbáno) 42
5.1 Spojení součástí ICON a ILA sloužící pro testování návrhu pomocí nástroje Chipscope Pro [12] 45
5.2 Nastavení ILA core spouštěcích parametrů 45
5.3 Nastavení ILA core spouštěcích parametrů 46
5.4 Možnost výběru signálů pro ILA z celého návrhu. 46
5.5 ILA core - připojení signálu *clk*. 46
5.6 ILA core - připojení signálu pro spouštění měření. 46
5.7 Průběh analýzy výstupů děličky frekvence přijímače nastavené na přenosovou rychlost 256000 bit/s. 47
5.8 Průběh analýzy vzorkování příchozích bitů přijímačem 47
5.9 Průběh analýzy vzorkování příchozích bitů přijímačem s vygenerovaným rušením 47
6.1 Blokové uspořádání laboratorní úlohy realizující spojení navrženého vysílače/přijímače s PC 49
6.2 Fotografie kompletního zapojení vývojové desky pro přehrávání zvukových dat, jehož návrh je uvedený v kapitole 4 50
A.1 Porovnání výstupu pulsně šířkové a sigma-delta modulace 59
B.1 Datový rámec UART 61
B.2 Stavový diagram řídicího stavového automatu přijímače 62
B.3 Blokové schéma laboratorní úlohy realizující spojení navrženého vysílače/přijímače s PC 63

Seznam tabulek

3.1 Tabulka standardních přenosových rychlostí [21], a chyby po jejich nastavení při výchozích velikostech registrů dělitele. Vstupní hodinovou frekvencí pro uvedené hodnoty je 100 MHz. 23
3.2 Hodnota paritního bitu podle nastavení vstupního signálu posuvného registru *par_opt* 25
3.3 Význam bitů výstupního registru *rx_data_status* indikujícího platnost přijatého datového rámce 40
3.4 Hodnoty adres nastavovacích registrů 41
4.1 Formát zpráv odesílaných programem v PC [3] 43
4.2 Formát potvrzovacích zpráv odesílaných vývojovou deskou [3] 43

Úvod

Tématem této bakalářské práce je návrh asynchronního sériového vysílače/přijímače (UART), implementace do hradlového pole a ověření jeho funkčnosti.

Cílem bakalářské práce je navrhnout UART v jazyce VHDL, který slouží pro popis hardware, a následně jej implementovat do hradlového pole. Návrh bude sloužit ke spojení již navrženého zvukového generátoru s PC. Navržený UART společně se zvukovým generátorem bude sloužit také jako laboratorní úloha kurzu „Logické obvody a systémy“.

V úvodní části práce je představen princip fungování UART a možnosti řízení datového toku. Dále je uveden příklad architektury UART, použité v mikrokontroléru M68HC11. Architektura UART M68HC11 využívá některé pokročilejší funkce, jako je například vzorkování příchozích bitů na šestnáctinásobné frekvenci přenosové rychlosti, detekce start bitu, apod.

V další části práce je přehled laboratorních úloh kurzu, které svým obsahem předcházejí navrhované úloze s UART. Jedná se zejména o úlohu č. 8 s konečným stavovým automatem a již připravovanou úlohu se zvukovým generátorem.

V kapitole 3 je detailněji popsán vlastní návrh asynchronního sériového vysílače/přijímače, jeho implementace do hradlového pole a simulace funkčnosti jeho jednotlivých částí. Na konci této kapitoly je popsaný postup nastavení parametrů sériového přenosu.

Kapitola 4 popisuje realizaci spojení UART s programovatelným zvukovým generátorem umožňující přehrávání hudebních dat odesílaných programem v PC.

Následně je krátce popsáno využití nástroje ChipScope Pro při testování návrhu. Tento nástroj umožňuje měřit průběhy signálů uvnitř FPGA a je součástí návrhového prostředí Xilinx ISE Webpack.

V poslední části práce je navrženo zadání studentských laboratorních úloh, které demonstrují funkci UART a také přehrávání skladeb pomocí zvukového generátoru.

1 Komunikační sběrnice UART

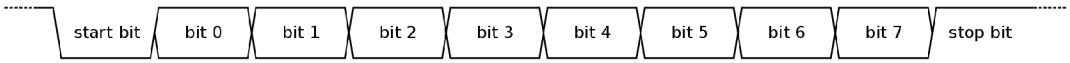
Tato kapitola popisuje princip fungování asynchronního sériového vysílače/přijímače. V kapitole je popsán základní datový rámec, využití vyrovnávací paměti a možnosti řízení toku dat. V poslední části kapitoly jsou popsány funkce asynchronního sériového vysílače/přijímače, vybraného mikrokontroléru M68HC11.

1.1 Základní popis

UART (Asynchronní sériový vysílač/přijímač) je dvouvodičová asynchronní sériová komunikační sběrnice, podporující režim full duplex. Protože je UART asynchronní sběrnice, jsou ke komunikaci dvou zařízení (vysílače a přijímače) postačující dva vodiče. Obecné označení těchto vodičů je **Tx** - odesílání dat a **Rx** - příjem dat. Pro synchronizaci má každé připojené zařízení svoje hodinové impulsy, které jsou nastaveny na určitou frekvenci, dle nastavení komunikační rychlosti. Jednotkou komunikační rychlosti je **bit/s** (někdy také označovaný jako **BAUD**).

1.2 Datový rámec UART

Na obrázku 1.1 je zobrazen průběh signálu na datovém vodiči.



Obr. 1.1: Datový rámec UART [15]

Přenos dat je vždy zahájen jedním start bitem s hodnotou log. 0, následují datové bity v pořadí od nejméně významného bitu (LSB) po nejvíce významný bit (MSB). Právě toto pořadí zaručuje, že při použití různých délek zpráv nemůže přenos začínat bitem jiné váhy. Datových bitů je přenášeno zpravidla 5 až 8. Pro potvrzení správnosti přenosu zprávy, může následovat sudý (even) nebo lichý (odd) paritní bit, který je nastaven do log. 0 nebo 1, podle počtu log. 1 ve zprávě. Tento způsob kontroly dat umožňuje detekci poškození jednoho bitu zprávy. Jako poslední je přenášen jeden, případně více stop bitů s hodnotou log. 1, které ukončují přenos zprávy. V případě, že některý ze stop bitů má hodnotu log. 0, není příchozí zpráva platná. V době, kdy není přenášena žádná zpráva, je datový vodič v log. 1. Tento stav na sběrnici se nazývá *idle line*. Právě hodnota log. 1 zajišťuje rozpoznatelnost stavu odpojení či přerušení datového vodiče od stavu *idle line* [15].

1.3 Vyrovnávací paměť

UART ve většině případů disponuje vyrovnávací pamětí typu FIFO (First In, First Out). Tato vyrovnávací paměť může být využita jako fronta vysílaných bytů. Především však bývá využívána jako buffer přijímaných dat (Receiver FIFO), jehož účelem je uchovat data do okamžiku přečtení dat nadřazeným systémem [13][14].

Často používanou vnitřní implementací vyrovnávací paměti je kruhové pole (Circular/Ring Buffer). Tato implementace je vhodná pro hardwarové aplikace. Velikost bufferu při použití kruhového pole je konstantní. Výhodou této implementace je nepotřebnost posuvu prvků v poli při zápisu. Řešením jsou dva ukazatele na pozici zápisu do pole a čtení z něho. Při zápisu do pole se příslušný ukazatel posouvá o 1 pozici dopředu. Při čtení se ukazatel posouvá stejným směrem. Do pole se tedy může zapisovat a číst, dokud ukazatel na zapisovací pozici není hned za ukazatelem pro čtení z pole (rozdíl mezi jejich pozicemi je délka celého pole). Tehdy je buffer naplněn [17].

1.4 Řízení toku dat

UART může obsahovat řízení datového toku, jehož účelem je pozastavení přenosu dat především v případě naplnění vyrovnávací paměti přijímače [18].

1.4.1 HW řízení toku dat

Hardwarové řízení toku dat bývá realizováno pomocí doplnění vysílače/přijímače výstupem **RTS** (Request To Send) a vstupem **CTS** (Clear To Send). Aktivace výstupu **RTS** přijímače indikuje naplnění vyrovnávací paměti přijímače. Výstup **RTS** přijímače bývá zapojený na vstup **CTS** vysílače protistrany, který tak může pozastavit či spustit vysílání. Přijímač takto generuje požadavek na odeslání dalších dat. Aktivní úroveň signálů **RTS** je log. 0, v případě mikrokontrolérů je tedy aktivní úroveň LOW. Při použití pro rozhraní RS232 je polarita signálů invertovaná, proto je aktivní úroveň tedy HIGH [18][13].

Rozhraní RS232 běžně využívá řídicí signály **DTR** (Data Terminal Ready) a **DSR** (Data Set Ready). Oba signály indikují, že zařízení může přijímat i vysílat data. Signál **DTR** je výstupním signálem zařízení typu **DTE** (Data Terminal Equipment), což je zařízení převádějící uživatelská data na signály komunikačního rozhraní, např.: PC. **DSR** je výstupní signál zařízení typu **DCE** (Data Circuit-terminating Equipment), které slouží ke komunikaci se zařízením **DTE** a následně zprostředkuje převod na jiné rozhraní. To může sloužit například k možnosti využít RS232 ke komunikaci dvou **DTE** zařízení na větší vzdálenosti [18][20][22][23].

1.4.2 SW řízení toku dat

Metoda softwarového řízení toku dat využívá pouze signálové vodiče **Tx** a **Rx**. Pokud dojde k naplnění vyrovnávací paměti přijímače daného zařízení, je zařízením vyslána řídicí zpráva **XOFF** oznamující nemožnost přijímat data. Pokud protistrana přijme zprávu **XOFF**, pozastaví vysílání. K obnově vysílání dojde až v okamžiku zpracování dat ve vyrovnávací paměti „přeplněného“ zařízení a následném vyslání řídicí zprávy **XON**. Řídicí znak **XOFF** je obecně reprezentován jako bajt s desítkovou hodnotou 19, **XON** s hodnotou 17 [19].

Výhodou softwarového řízení toku dat je nepotřebnost dalších vodičů. Avšak nevýhodou je prodleva při vysílání pozastavovacího znaku **XOFF**, a také může být pozdrženo jeho zpracování, vedoucí k pozdnímu pozastavení vysílání [19][18].

1.5 Architektura UART mikrokontroléru M68HC11

Rodina mikrokontrolérů M68HC11 od firmy Freescale využívá k sériové komunikaci rozhraní UART označené jako SCI (Serial Communication Interface). Toto rozhraní podporuje *full duplex*. Využívá standardní formát zprávy (jeden start bit, 8 datových bitů případně i 9. bit pro využití parity a jeden stop bit) [6].

V této kapitole jsou představeny některé vlastnosti, kterými tento UART disponuje.

1.5.1 Funkce Send Break

Mikrokontrolér M68HC11 disponuje funkcí *Send Break* pro přerušení přenosu. Vysílač M68HC11 může vystavit na datový vodič **TxD** trvalou hodnotu log. 0 a to po dobu celistvého počtu datových rámců. Funkce je aktivována nastavením bitu **SBK** registru **SCCR2** (SCI Control Register 2) do log. 1. Pokud se hodnota bitu **SBK** změní na log. 0, je dokončen přenos celého přerušovacího znaku a až poté může přenos dat pokračovat [6].

1.5.2 Funkce Queued Idle Character

Vysílač mikrokontroléru M68HC11 může na datový vodič **TxD** vystavit log. 1 po dobu celistvého počtu datových rámců. Této funkce se zpravidla využívá jako počáteční zprávy při změně bitu **TE** (Transmit Enable) registru **SCCR2** (SCI Control Register 2) z log. 0 do log. 1. Jeden takto vyslaný rámec, použitý k zahájení přenosu, se nazývá preambule. Z pohledu přijímače protistrany je na vstupu **RxD** hodnota

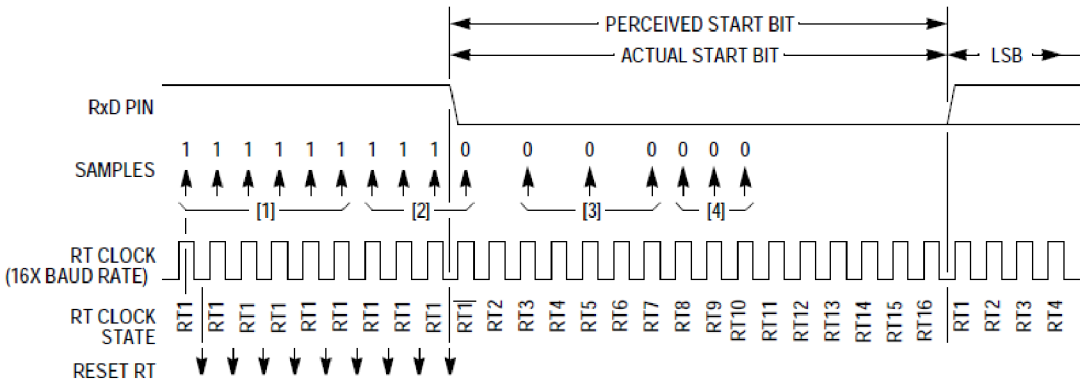
log. 1 po dobu celistvého počtu datových rámců, při přerušení přenosu. Po obnovení přenosu, je tedy start bit prvního datového rámce detekován vždy v očekávaný okamžik [6].

1.5.3 Vzorkování dat přijímačem

Přijímač mikrokontroléru M68HC11 využívá vzorkovací frekvence představující šestnáctinásobek frekvence přenosové rychlosti. Tento hodinový signál je označen jako **RT clock**, přičemž periody tohoto hodinového signálu jsou označeny čísly 1 až 16 (RT1 je perioda na začátku přijímaného bitu a RT16 je poslední perioda na konci přijímaného bitu). Jednotlivé bity jsou vzorkovány v čase tří period RT uprostřed intervalu doby jednoho bitu (RT8, RT9 a RT10). Logická úroveň přijímaného bitu je vyhodnocena podle úrovně většiny těchto vzorků [6].

1.5.4 Detekce start bitu

Start bit je detekován jako první vzorek s úrovní log. 0 na pinu **RxD** a je považován za RT1. Při správně detekovaném start bitu předcházejí tomuto vzorku tři vzorky na úrovni log. 1. Dále je pak hodnota start bitu verifikována pomocí vzorků RT3, RT5 a RT7. Jako další ověření slouží vzorky RT8 až RT10, používané pro každý bit. Nicméně i přesto, že dva z těchto bitů mají hodnotu log. 1, je start bit validní. Detekce start bitu v ideálním případě je zobrazena na obrázku 1.2. Výrobce definuje také několik případů detekce start bitu s indikací šumu. V těchto případech je tedy nastavený příznakový bit **NF** (Noise Flag) [6].



Obr. 1.2: Detekce start bitu přijímačem M68HC11 – ideální případ [6]

2 Zadání vybraných úloh kurzu BPC-LOS

Tato kapitola popisuje zadání několika laboratorních úloh kurzu „Logické obvody a systémy“, jejichž náplň souvisí s některými prvky návrhu UART.

2.1 Úloha č. 6 - Čítače a rozšířená dělička frekvence

Úloha č. 6 kurzu BPC-LOS spadá do oblasti úloh se sekvenčními logickými obvody. Zabývá se návrhem synchronních a asynchronních binárních čítačů, jejichž rozdíl je v zadání úlohy objasněn. Prvním úkolem je rozšířit děličku frekvence využívanou v předcházejících úlohách o vstup asynchronního resetu, vstup povolení čítání a povolovací výstup **CEO** (Clock Enable Out). Dalším úkolem je implementace BCD čítače asynchronním i synchronním způsobem. Asynchronní čítač využívá jako hodinový signál výstupní podělenou frekvenci děličky. Synchronní čítač využívá stejné vstupní hodinové frekvence jako dělička, a je doplněn o vstup povolující čítání **CE** (Clock Enable). Povolovací impuls pro čítání generuje upravená dělička svým výstupem **CEO** [4].

2.2 Úloha č. 8 - Konečný stavový automat

Úloha č. 8 se zabývá možnostmi odstranění zákmitů a synchronizace vstupu z tlačítka. Obvod pro odstranění zákmitů se nazývá *debouncer* a je možné je realizovat analogově (dolní propusť) i digitálně (čítače, posuvné registry, stavové automaty). V úloze je zadáno navrhnout digitální debouncer s využitím konečného stavového automatu Mooreova typu. Stavový automat obsahuje 6 stavů, výstup entity se nastaví do log. 1, až ve chvíli, kdy se stavový automat nachází ve stavu 4, což nastane pouze při 4 po sobě navzorkovaných hodnot log. 1 na vstupu entity. Tímto je zajištěno odstranění zákmitu při přechodu úrovně na vstupu z log. 0 do log. 1 (stisk tlačítka). Při opačném přechodu (uvolnění tlačítka) je výstup navrácen do log. 0 hned při prvním vzorku s úrovní log. 0. Pokud bychom chtěli výstup držet na úrovni log. 1, dokud nebude po uvolnění tlačítka na vstupu log. 0 po dobu několika vzorků. Bylo by třeba navýšit počet stavů a vhodně uzpůsobit přechody mezi nimi [5].

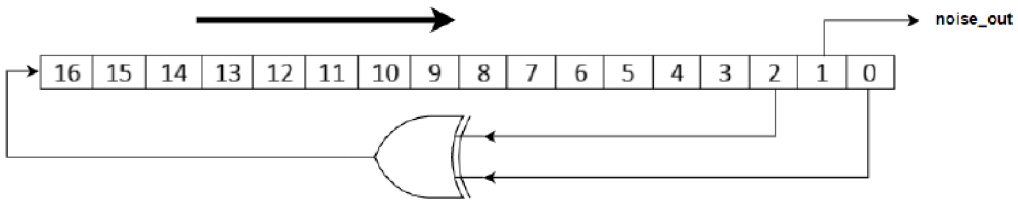
Vypracování této úlohy by mělo být nácvikem pro mnou připravovanou úlohu s UART, především z hlediska pochopení funkce stavového automatu a jeho implementace do hradlového pole, který je při návrhu UART využitý. Dále je v zadání této úlohy principiální schéma obvodu pro odstranění zákmitů tlačítka na hradlové úrovni, kde je využitý vstupní obvod dvojitého vzorkovače určený k potlačení metastabilního děje na vstupu do obvodu.

2.3 Připravovaná úloha se zvukovým generátorem

V připravované úloze s programovatelným zvukovým generátorem (**PSG**) je úkolem realizovat generátor frekvence a generátor šumu. Studenti budou mít k dispozici deklarace entit obou komponent [3].

Generátor frekvence má být ze zadání realizován jako synchronní binární čítač, který je již obsažen v úloze č. 6 kurzu. Oproti úloze č. 6, kde je úkolem navrhnout BCD čítač, je požadován pro generátor frekvence pouze binární čítač. Dělitel frekvence bude reprezentován 12-bitovým vektorem, který bude vytvořen ze dvou vstupních vektorů *coarse_in* (bity 11 až 8) a *fine_in* (bity 7 až 0). Čítání čítače bude povoleno vstupním signálem *CE* (Clock Enable). Výstup komponenty označený jako *frequency_out*, bude podělená frekvence vstupního signálu *CE*. Realizace bude tedy velmi podobná realizaci děličky signálu známé z několika předchozích úloh. Komponenta bude také obsahovat vstup asynchronního resetu *rst* [3].

Generátor šumu má být realizován jako 17-bitový lineární zpětnovazební čítač fibonacciho typu (LFSR). Komponenta bude obsahovat vstup asynchronního resetu *rst*. Čítání bude také povolováno vstupním signálem *CE*. Interval čítání bude určen vstupním 5-bitovým vektorem *R6_in*. Jako výchozí hodnota lineárního zpětnovazebního čítače bude hodnota X"1FFFF". Výstup komponenty *noise_out* je bit 1 zpětnovazebního čítače jak je zobrazeno na obrázku [3].



Obr. 2.1: Lineární zpětnovazební čítač použitý v generátoru šumu [3]

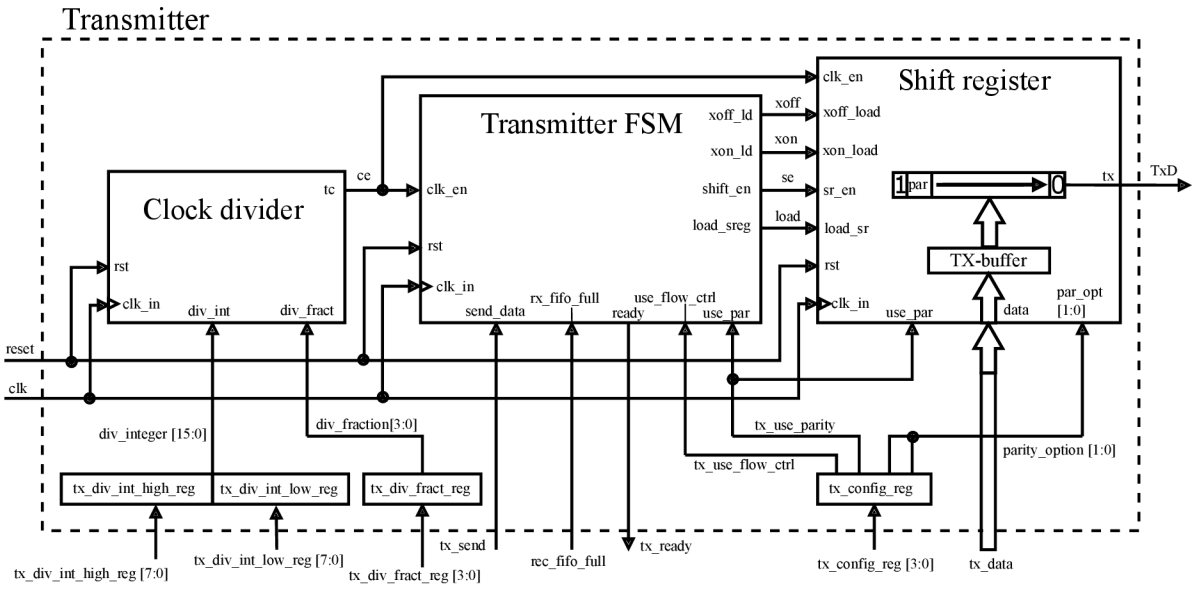
Tato úloha seznámí studenty s principem generování zvuku pomocí PSG. S touto úlohou bude později spojena připravovaná úloha s UART tak, aby bylo možné přehrávat skladby přímo z PC.

3 Vnitřní blokové schéma navrženého UART a simulace jednotlivých bloků

Navržený UART je rozdělen na dva hlavní bloky vysílač (Transmitter) a přijímač (Receiver). Oba bloky jsou složeny z několika dílčích entit, které byly samostatně navrženy a následně realizovány v jazyce VHDL. Každá z entit je realizována *behaviorálním* stylem popisu architektury. Sestavení obou hlavních bloků z jednotlivých komponent je realizováno *strukturálním* stylem popisu propojujícím jednotlivé entity podle navržených blokových schémat. Implementace návrhu a simulace dílčích entit byla provedena v návrhovém prostředí *Xilinx ISE Webpack*.

3.1 Vysílač UART

Navržený vysílač je sestaven ze tří dílčích bloků. Blokové schéma vysílače je zobrazeno na obrázku 3.1. Hlavním blokem je řídicí stavový automat (Transmitter FSM), který ovládá výstupní posuvný registr signály *load* (load shiftregister), *se* (Shift Enable), *xoff* a *xon* . Frekvence posuvu je určena výstupní frekvencí děličky frekvence (Clock divider).



Obr. 3.1: Blokové schéma vysílače

3.1.1 Dělička hodinového signálu

Blok děličky frekvence hodinového signálu vychází z laboratorní úlohy č. 6 kurzu Logické obvody a systémy. Pro vysílač je do základní děličky implementován vstup synchronního resetu *rst*. Účelem této děličky frekvence je generovat povolovací signál *tc* (Terminal Count), který bude povolovat vysílání jednotlivých bitů a to zvolenou vysílací frekvencí (Baudrate). Vstupní frekvencí je signál *clk_in*, tedy hlavní hodinová frekvence vývojové desky. Dělička je založena na podobném principu jako baudrate generátor UART chipu MAX3108, kde je nazývaný jako **Fractional baudrate generator** [14]. Díky tomuto principu je možné nastavení téměř libovolně přesnou přenosovou rychlost. Dělička tedy obsahuje dva čítače pro nastavení přenosové rychlosti. První čítač je binární a slouží k dělení vstupního hodinového signálu celou částí čísla vypočítaného dělitele, které je nastaveno vstupním vektorem *div_int*. Druhý čítač zajišťuje dělení vstupního hodinového signálu desetinnou částí čísla vypočítaného dělitele. Tento čítač funguje na zcela odlišném principu než binární čítač pro dělení celým číslem dělitele, a to na principu **sigma-delta** digitálně-analogového převodníku, jehož funkcionalita je interpretována v [2]. Sigma-delta čítač oproti binárnímu čítači neinkrementuje pouze číslo 1, ale číslo nastavené vstupním vektorem *div_fract*. Inkrementace tohoto čítače se provede při každém napočítání hodnoty celého čísla dělitele prvního čítače. Šířka druhého čítače je o 1 vyšší než šířka vstupního vektoru *div_fract*. Výstupem tohoto čítače je MSB, který je použitý jako přenos čítače. Při každém přetečení čítače se interval prvního čítače zkrátí o 1. Výstupem těchto dvou čítačů je impuls signálu *terminal_count_16* (dopočítání prvního čítače). Interval prvního čítače se tedy mění o jednu periodu vstupního hodinového signálu v poměru daném vstupním vektorem *div_fract*.

Pro získání výstupního signálu *tc* je frekvence pulsů signálu *terminal_count_16* vydělena šestnácti pomocí dalšího binárního čítače. Důvodem dalšího dělení frekvence je vzorkování vstupních dat přijímače na šestnáctinásobné frekvenci, jak je blíže popsáno v kapitole 3.2, aby bylo možné přenosovou rychlost celého UART nastavovat stejnými hodnotami.

Výpočet dělitele pro nastavení přenosové rychlosti

Dělička je programovatelná nastavením vstupních signálů *div_int* a *div_fract*. Výpočet těchto hodnot je uveden na následujícím příkladu pro přenosovou rychlost 19200 bit/s:

$$DIV = \frac{f_{clk}}{16 \cdot BAUDRATE} = \frac{100 \cdot 10^6}{16 \cdot 19200} = 325.52\ [-, \mathrm{Hz}, \mathrm{bit/s}] \tag{3.1}$$

$$div\_int = 325\ [-] \tag{3.2}$$

$$div\_fract = 2^{DIV\_FRACT\_BITS} \cdot 0.52 = 2^4 \cdot 0.52 = 8\ [-] \tag{3.3}$$

Analýza přesnosti nastavení přenosové rychlosti

Velikosti registrů pro nastavení přenosové rychlosti je možné parametrizovat nastavením generických parametrů entity. Proto je zapotřebí určit optimální velikost těchto registrů. Registr pro nastavení celé části dělitele *div_int*, je ideální využít 16 bitový. Jelikož velikost tohoto registru ovlivňuje rozsah možných nastavení přenosových rychlostí, především pak nejmenší možnou přenosovou rychlost. Parametr velikosti tohoto vektoru *DIV_INT_BITS* je generický, proto je možné ho před syntézou návrhu změnit. Výchozí hodnota tohoto parametru 16 bitů byla zvolena pro možnost nastavení všech standardních přenosových rychlostí uvedených v tabulce 3.1. Výpočet minimální přenosové rychlosti dle velikosti registru definuje rovnice 3.4. Pro často používané rychlosti od 9600 bit/s je však postačující šířka registru 10 bitů. V případě, že není třeba využívat nižší přenosové rychlosti, je tak možné přenastavením hodnoty parametru *DIV_INT_BITS* zredukovat využití prostředků FPGA.

$$\begin{aligned} MIN\_BAUDRATE &= \frac{f_{clk}}{16 \cdot MAX\_DIV} = \frac{100 \cdot 10^6}{16 \cdot (2^{DIV\_INT\_BITS} - 1)} = \\ &= \frac{100 \cdot 10^6}{16 \cdot (2^{16} - 1)} \approx 96\ [\mathrm{bit/s, Hz}, -] \end{aligned} \tag{3.4}$$

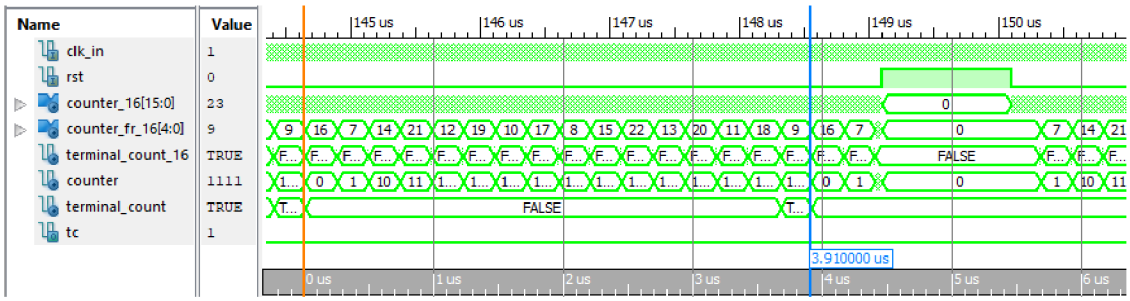
Velikost registru pro nastavení desetinné části dělitele *div_fract* je volena na základě požadované přesnosti nastavení přenosové rychlosti. Pro nastavení široké škály přenosových rychlostí je postačující velikost registru 4 bity. Nejmenší možná změna hodnoty dělitele je tedy $2^{-4} = 0,0625$. Maximální chyba nastavení standardních přenosových rychlostí je vypočítána v tabulce 3.1. V pravém sloupci této tabulky je vypočítaný poměr doby maximální možné chyby k době trvání odesílání jednoho bitu, po odeslání jednoho datového rámce včetně paritního bitu, tedy po odeslání 11-ti bitů.

Tab. 3.1: Tabulka standardních přenosových rychlostí [21], a chyby po jejich nastavení při výchozích velikostech registrů dělitele. Vstupní hodinovou frekvencí pro uvedené hodnoty je 100 MHz.

Požadovaná rychlost	Nastavená rychlost	Maximální chyba	Poměrná část bitu
bit/s	bit/s	%	%
110	110,0000	$-1 \cdot 10^{-5}$	$11 \cdot 10^{-5}$
134,5	134,5001	$-5,7 \cdot 10^{-5}$	$62,7 \cdot 10^{-5}$
150	149,9999	$-5 \cdot 10^{-5}$	$55 \cdot 10^{-5}$
300	300,0003	$1 \cdot 10^{-4}$	$1,1 \cdot 10^{-3}$
600	599,9988	$-2 \cdot 10^{-4}$	$2,2 \cdot 10^{-3}$
1200	1200,0048	$4 \cdot 10^{-4}$	$4,4 \cdot 10^{-3}$
1800	1799,9808	$-8 \cdot 10^{-4}$	$8,8 \cdot 10^{-3}$
2400	2399,9808	$-8 \cdot 10^{-4}$	$8,8 \cdot 10^{-3}$
4800	4800,0768	$1,6 \cdot 10^{-3}$	$17,6 \cdot 10^{-3}$
7200	7199,9424	$-8 \cdot 10^{-4}$	$8,8 \cdot 10^{-3}$
9600	9599,9424	$-3,2 \cdot 10^{-3}$	$35,2 \cdot 10^{-3}$
14400	14400,9217	$6,4 \cdot 10^{-3}$	$70,405 \cdot 10^{-3}$
19200	19201,2289	$6,4 \cdot 10^{-3}$	$70,405 \cdot 10^{-3}$
38400	38402,4578	$6,4 \cdot 10^{-3}$	$70,405 \cdot 10^{-3}$
56000	55991,0414	$-16 \cdot 10^{-3}$	$176 \cdot 10^{-3}$
57600	57603,6866	$6,4 \cdot 10^{-3}$	$70,405 \cdot 10^{-3}$
115200	115207,3733	$6,4 \cdot 10^{-3}$	$70,405 \cdot 10^{-3}$
128000	128040,9731	$32 \cdot 10^{-3}$	$352 \cdot 10^{-3}$
256000	255754,4757	$-96 \cdot 10^{-3}$	1,054

Simulace děličky frekvence přijímače

Na obrázku 3.2 je zobrazena časová simulace děličky, která generuje výstupní povolovací signál *tc* pro nastavenou přenosovou rychlost 256000 bit/s. Tato přenosová rychlost byla pro simulaci zvolena z důvodu, že hodnota nastavovacího registru pro desetinnou část dělitele *div_fract* je 7, což je z hlediska znázornění funkce děličky z výčtu standardních hodnot nejzajímavější. Z průběhu signálu desetinného čítače *counter_fr_16* je vidět inkrementace o nastavené číslo 7. Kurzory na obrázku zvýrazňují okamžik pulsů výstupního signálu *tc*. Ve spodní části obrázku je pravítko ukazující hodnotu doby trvání jednoho bitu, tedy $1/256000 = 3,91\ \mu s$. V pravé části obrázku je vygenerovaný reset, který vynuluje čítače a po jeho nastavení zpět na hodnotu log. 0 se opět obnovuje chod všech čítačů.



Obr. 3.2: Simulace děličky frekvence hodinového signálu vysílače

3.1.2 Posuvný registr vysílače

Blok posuvného registru vysílače slouží k vysílání dat, které jsou na jeho vstupu s označením *data*. Uvnitř entity jsou nadefinovány dva registry *shift_reg* a *tx_buffer*. Registr *shift_reg* představuje vlastní posuvný registr, má velikost 10 bitů (start bit, stop bit a 8 datových bitů). Registr *tx_buffer*, o velikosti 8 bitů, představuje buffer zajišťující možnost načtení dalších dat pro následující vysílání. Přenos datových bitů z registru *tx_buffer* do registru *shift_reg* proběhne vždy na náběžnou hranu hodinového signálu *clk*, pokud je vstupní signál *load_sr* (Load Shift Register) v log. 1. Frekvence posuvu je řízena vstupním signálem *clk_en* (Clock Enable), generovaným děličkou frekvence popsané v předchozí podkapitole 3.1.1. Posuv je povolen při aktivní úrovni vstupního signálu *sr_en* (Shift Enable). Vstupní signál synchronního resetu *rst* zajistí naplnění posuvného registru *shift_reg* hodnotami log. 1, což odpovídá klidové úrovni sériové linky.

Generování paritního bitu

Posuvný registr je doplněn o vstup *use_par* (Use Parity), který určuje zda má být vysílán i paritní bit. Pokud je tento vstupní signál na úrovni log. 1, je paritní bit vygenerován a odeslán po posledním odesílaném datovém bitu MSB. K volbě druhu paritního bitu slouží vstupní signál typu vektor *par_opt* (Parity Option). Dostupné jsou tedy 4 možné typy paritního bitu (Even, Odd, Space, Mark), jejichž odpovídající hodnoty vstupu *par_opt* jsou uvedeny v tabulce 3.2. Zvolení paritního bitu typu Mark, lze také považovat za přenos rámce se dvěma stop bity.

Jako generátor parity ve VHDL popisu posuvného registru byla nadefinována a použita funkce *f_parity_generator*, která je součástí uživatelského knihovního balíku *parity.vhd*. Funkci jsou předávány parametry *par_opt* a *tx_buffer* a vrací vygenerovanou hodnotu parity, kterou ukládá do interního signálu *parity*. Pro výpočet sudého či lichého počtu log. 1 je použita log. funkce XOR nad celým vektorem s daty. Funkce je volána v procesu, při každé změně vstupního hodinového signálu *clk_in*. Hodnota

parity je tedy neustále přepočítávána podle aktuálních dat v bufferu a v okamžiku přesunu dat z bufferu do posuvného registru je přesunuta společně s těmito daty, za předpokladu aktivního signálu *use_par*.

Tab. 3.2: Hodnota paritního bitu podle nastavení vstupního signálu posuvného registru *par_opt*

Hodnota *par_opt*	Typ parity	Hodnota parity
00	Even	Sudý počet log. 1 v rámci
01	Odd	Lichý počet log. 1 v rámci
10	Space	0
11	Mark	1

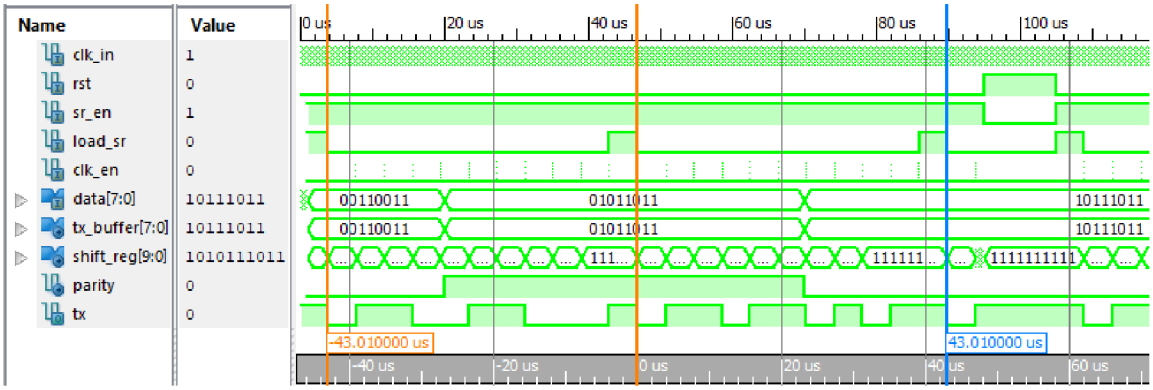
Implementace softwarového řízení datového toku

Posuvný registr musí v případě požadavku přednostně odeslat pozastavující znak XOFF a znak XON obnovující přenos, jak je popsáno v podkapitole 1.4.2. Odesílání těchto znaků je řízeno vstupními signály *xoff_load* a *xon_load*. V případě, že je některý z nich aktivní, je do posuvného registru přesunuta hodnota daného znaku namísto vstupních dat.

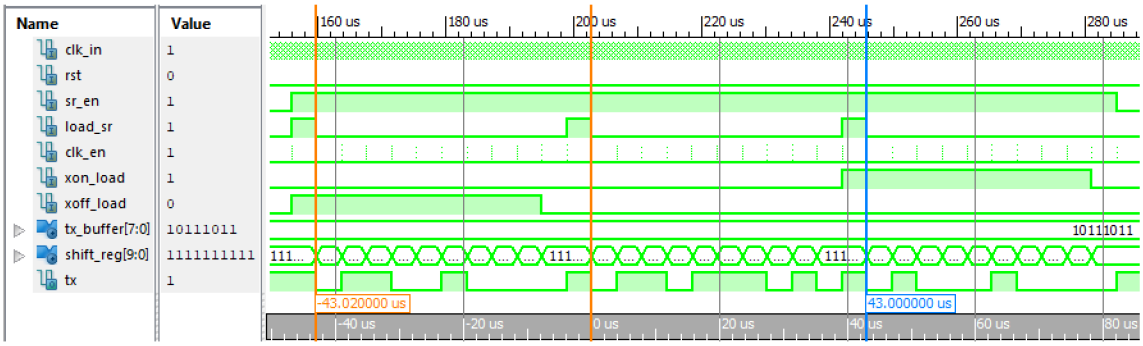
Simulace posuvného registru

Na obrázku 3.3 je zobrazena simulace bloku posuvného registru vysílače. V simulaci je vidět pomyslné odeslání dvou datových rámců. Na tomto obrázku je pro simulaci povoleno odesílání paritního bitu parity a je zvolena sudá parita. Počet log. 1 v prvním datovém rámci je sudý a proto má odesílaný paritní bit hodnotu log. 0. Data následujícího datového rámce mají lichý počet log. 1 a vygenerovaná hodnota paritního bitu je log. 1, tedy doplnění do sudého počtu log. 1 v celém datovém rámci. Při odesílání dalšího rámce je vygenerován vstupní signál *rst*, čímž je způsobena inicializace posuvného registru *shift_reg* na hodnotu log. 1 všech bitů a datový výstup *tx* je nastaven rovněž na hodnotu log. 1, tedy na klidovou úroveň sériové linky.

Na obrázku 3.4 je zobrazena simulace bloku posuvného registru při odesílání znaků XON/XOFF. Pozastavovací znak XOFF je odeslán při aktivním vstupním signálu *xoff_load* a okamžik jeho přesunutí do posuvného registru je označen prvním oranžovým kurzorem. Dále je odeslán požadovaný datový rámec, viz druhý oranžový kurzor. Modrým kurzorem je označeno odeslání obnovovacího znaku XON.



Obr. 3.3: Simulace bloku posuvného registru vysílače s odesláním dvou datových rámců a vygenerovaným signálem *reset*. V simulaci je nastavena sudá parita.



Obr. 3.4: Simulace bloku posuvného registru vysílače s odesláním pozastavovacího znaku XOFF a obnovovacího znaku XON, softwarového řízení datového toku

3.1.3 Řídicí stavový automat vysílače

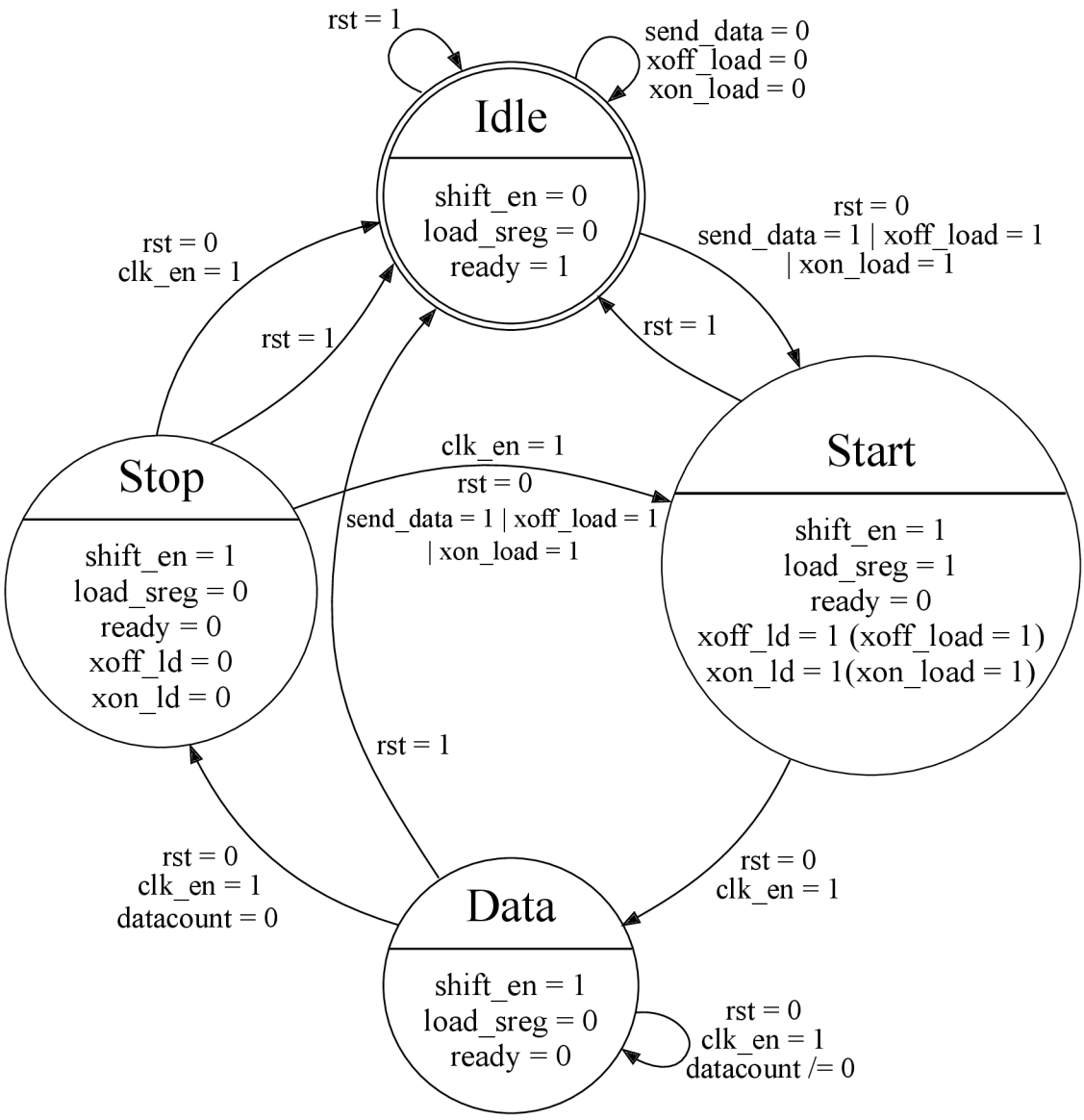
Jádrem vysílače je řídicí blok s konečným stavovým automatem Moorova typu. Stavový automat obsahuje stavy **Idle**, **Start**, **Data** a **Stop**. Chování stavového automatu je popsáno pomocí stavového diagramu, zobrazeného na obrázku 3.5. Všechny přechody mezi stavy stavového automatu jsou plně synchronní se vstupním hodinovým signálem *clk_in*. Vstup synchronního resetu *rst* přednostně navrací stavový automat do počátečního stavu **Idle**, ve kterém se čeká na náběžnou hranu hodinového signálu *clk* se současně aktivním vstupním signálem *send_data* zahajujícím vysílání, či jedním z interních signálů *xoff_load* nebo *xon_load*. Výstupní signál *ready* je aktivní pouze ve stavu **Idle** a podává tak informaci o připravenosti vysílače na vysílání. Ve stavu **Start** se nastaví na výstupy *load_sreg* a *shift_en* log. 1. Přechody mezi ostatními stavy nastanou pouze v případě, kdy je vstupní signál *clk_en* v log. 1, tedy dle zvolené vysílací frekvence generované děličkou frekvence. Stav **Data**

obsahuje čítač odeslaných bitů, kdy se postupně s frekvencí vstupu *clk_en* dekrementuje vnitřní signál *datacount*, přičemž počáteční hodnota signálu *datacount* pro odeslání jednoho datového rámce je nastavena na hodnotu 7 nebo 8, podle požadavku na odesílání paritního bitu, určeném hodnotou vstupního signálu *use_par*. Po skončení odpočtu přechází stavový automat do stavu **Stop**. V tomto stavu se již výstupy nijak nemění, slouží pouze jako stav reprezentující odesílání stop bitu posuvným registrem. Pro zajištění možnosti okamžitého odesílání dalšího datového rámce, přechází automat do stavu **Start** dle požadavku na odeslání.

Realizace entity stavového automatu ve VHDL je tvořena třemi procesy, které reprezentují jednotlivé části obecného popisu stavového automatu. Proces vstupního kombinačního obvodu je implementován sekvenčním příkazem CASE-WHEN, určujícím následující stav automatu podle kombinací vstupů znázorněných ve stavovém diagramu na obrázku 3.5. Proces reprezentující registrovou část zajišťuje synchronní přechod mezi aktuálním stavem a nadcházejícím stavem definovaným vstupním kombinačním obvodem. Dále je v tomto procesu nadefinována dekrementace a inicializace vnitřního signálu *datacount*. V tomto procesu se provádí také zápis na výstupní signály *xoff_ld* a *xon_ld*, jak je blíže popsáno v následujícím odstavci. Třetí proces realizuje výstupní kombinační obvod také pomocí sekvenčního příkazu CASE-WHEN, který pouze nastavuje výstupní signály podle aktuálního stavu a při jeho změně. Zdrojový soubor s popisem entity je součástí elektronické přílohy.

Realizace softwarového řízení datového toku

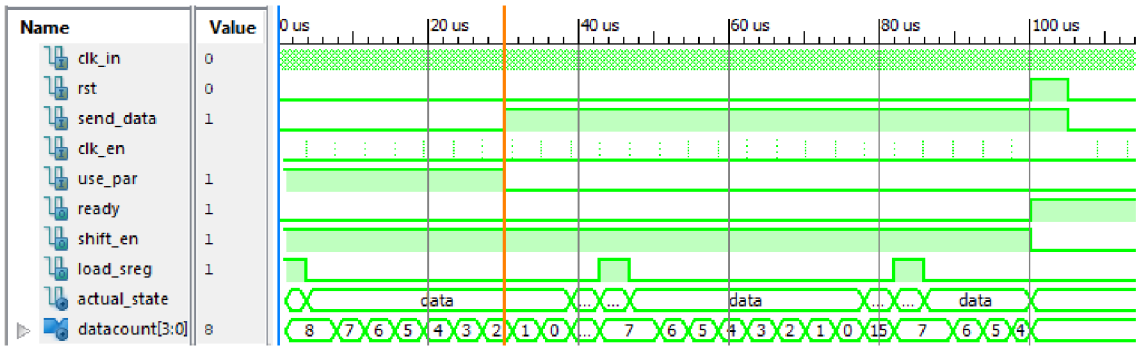
Požadavek na povolení funkce SW řízení toku dat, je udáván hodnotou vstupního signálu *use_flow_ctrl*. Znaky XOFF/XON jsou odeslány při změně vstupního signálu *rx_fifo_full*, který nese informaci o zaplnění bufferu přijímače. V případě změny signálu na hodnotu log. 1 je nastaven vnitřní signál *xoff_send*. V opačném případě (dostatečném vyprázdnění bufferu) je nastaven vnitřní signál *xon_send*. Tyto vnitřní signály spustí vysílání datového rámce a ve stavu **Start** aktivují příslušné výstupní signály *xoff_ld* nebo *xon_ld* (XOFF/XON Load). Tyto signály se resetují po odeslání příslušné zprávy, aby byla odeslána pouze jednou.



Obr. 3.5: Stavový diagram řídicího stavového automatu vysílače

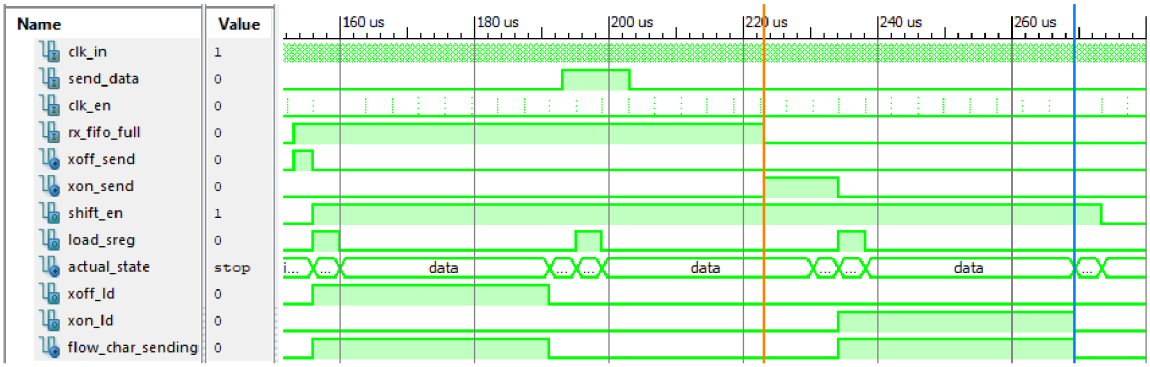
Simulace stavového automatu vysílače

Na obrázku 3.6 je výstup simulace řídicího stavového automatu vysílače. Nejdříve je nasimulováno odeslání datového rámce s paritním bitem. Modrým kurzorem je označen puls vstupního signálu *send_data* s šířkou jedné periody signálu *clk_in*. Během odesílání tohoto rámce je vypnuto odesílání paritního bitu v důsledku deaktivace vstupního signálu *use_par* označené oranžovým kurzorem. Na začátku odesílání další zprávy je signál *datacount* nastaven na hodnotu 7, za účelem odeslání nižšího počtu bitů oproti první zprávě. Během odesílání dalšího rámce je vygenerován signál reset.



Obr. 3.6: Simulace bloku řídicího stavového automatu vysílače s odesíláním datového rámce s paritním bitem, bez něj a provedením resetu

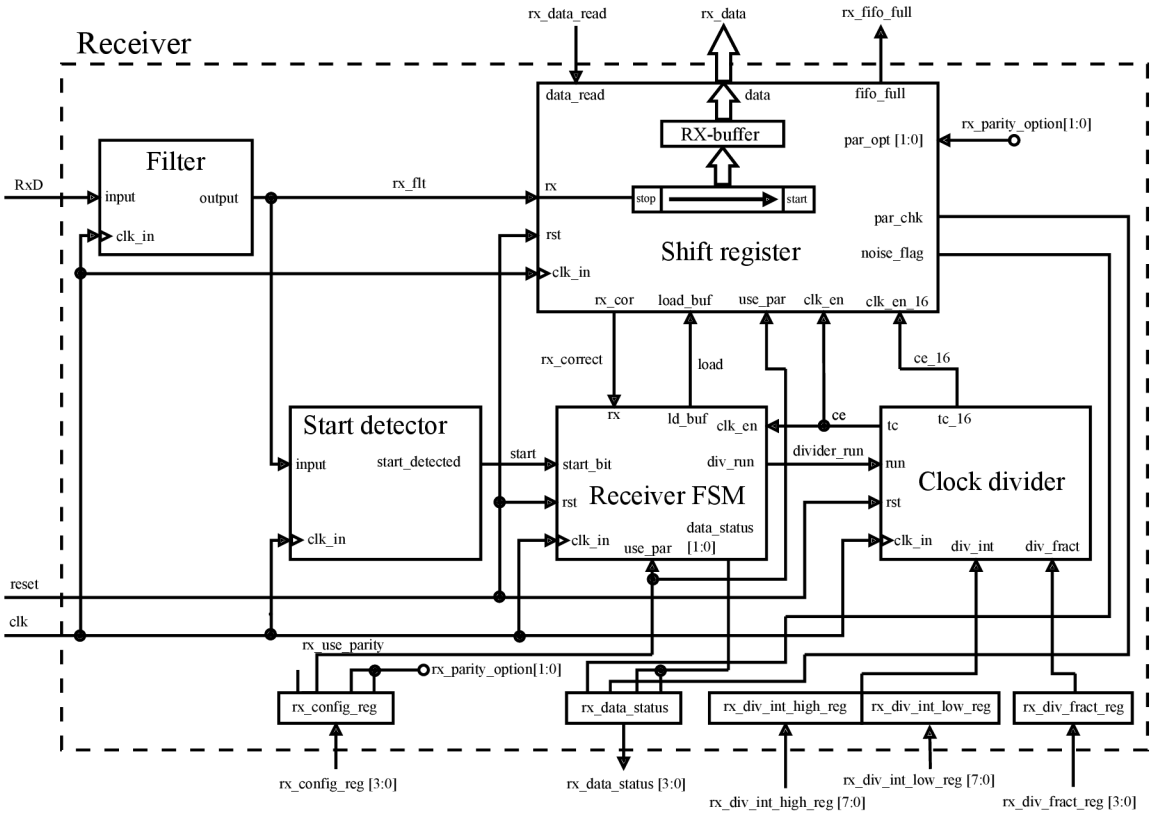
Simulace znázorňující vygenerování signálů řídicích odesílání znaků SW řízení toku dat je na obrázku 3.7. V první části simulace je změněn vstupní signál *rx_fifo_full* a je vygenerován interní signál *xoff_send*, který spustí odesílání zprávy. Výstupní signál *xoff_ld*, nastavený na hodnotu log. 1 je následně deaktivován ve stavu **Stop**. Pro zajištění odeslání znaku pouze jednou je použitý pomocný vnitřní signál *flow_char_sending*, který deaktivuje signál *xoff_send*. V další části simulace jsou odeslána aktuální data a po vyprázdnění bufferu přijímače znak XON stejným způsobem.



Obr. 3.7: Simulace bloku řídicího stavového automatu vysílače s odesláním znaků softwarového řízení toku dat

3.2 Přijímač UART

Navržený přijímač je sestaven z 5 dílčích bloků. Oproti vysílači obsahuje navíc vstupní filtr a detektor start bitu. Ostatní bloky jsou obdobou bloků vysílače, popsaného v kapitole 3.1, s některými úpravami. Blokové schéma celého přijímače je znázorněno na obrázku 3.8.



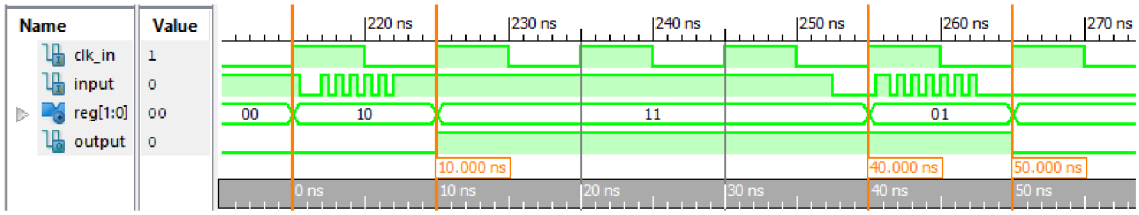
Obr. 3.8: Blokové schéma přijímače

3.2.1 Filtr metastabilního děje

Blok **Filter** je v architektuře přijímače použitý za účelem zamezení možného šíření *metastabilního stavu* na vstupu obvodu, dále do obvodu. Blok také zaručí podmínky povinných dob předstihu a přesahu, definujících dobu neměnnosti vstupního signálu vzhledem k hodinovému signálu. Princip funkce bloku je postaven na principu *dvojitého vzorkovače*. Blok je implementován jako posuvný registr. Entita je generická a je tedy možné nastavovat počet bitů posuvného registru při vkládání entity ve strukturálním popisu nadřazené entity. Pokud je nastavený počet bitů posuvného registru vyšší, je výstupní signál více zpožděný od vstupního. Výhodu vyššího počtu

bitů je však možnost odstranění také metastabilních dějů delších, než je perioda vstupního hodinového signálu filtru.

Simulace bloku je zobrazena na obrázku 3.9. V levé části obrázku je vidět nasimulovaný metastabilní děj na vstupu *input*, který je kratší než perioda hodinového signálu filtru *clk_in*. Dvěma kurzory jsou znázorněny náběžné hrany hodinového signálu, kdy má vstupní signál úroveň log. 1. Simulovaný metastabilní děj tedy neovlivní výstup *output* filtru, který při druhé náběžné hraně hodinového signálu přechází ze stavu log. 0 do log. 1. Obdobně je otestován přechod vstupního signálu z úrovně log. 1 do log. 0, jak je vymezeno dvěma kurzory v pravé části obrázku.

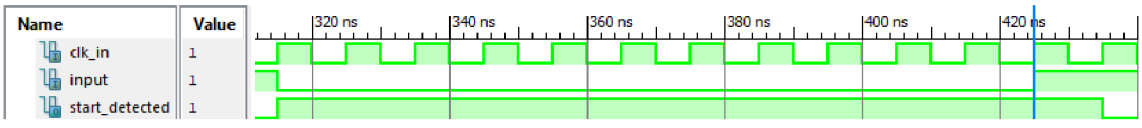


Obr. 3.9: Simulace dvou metastabilních dějů na vstupu bloku Filter, použitého v přijímači

3.2.2 Detektor start bitu

Blok **Start detector** je realizován jako detektor sestupné hrany vstupního signálu. Při zaznamenání sestupné hrany vstupního signálu *input* je nastaven výstupní signál *start_detected* na hodnotu log. 1.

Simulace detektoru start bitu je zobrazena na obrázku 3.10, přičemž je nutné poznamenat, že při propojení entity s filterem, jak je vidět ve schématu přijímače na obrázku 3.8, je výstupní signál *start_detected* zpožděný o jednu periodu hodinového signálu *clk_in*.



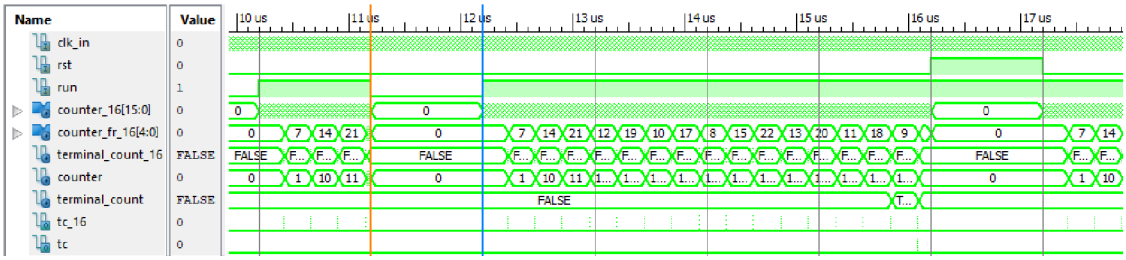
Obr. 3.10: Simulace funkce bloku Start detector

3.2.3 Dělička hodinového signálu

Dělička frekvence hodinového signálu pro přijímač je navržena jako modifikace děličky použité pro generování frekvence dle přenosové rychlosti vysílače. Dělička je doplněna o vstup *run* sloužící pro spouštění děličky pouze v době, kdy přijímač přijímá data. Dále je dělička doplněna o výstup *tc_16*, který generuje 16 pulsů během intervalu mezi dvěma pulsy na výstupu *tc*. Tento signál slouží k možnosti vzorkovat příchozí bity na šestnáctinásobné frekvenci. Signál je generován při dokončení intervalu prvního čítače, tedy při aktivním signálu *terminal_count_16*.

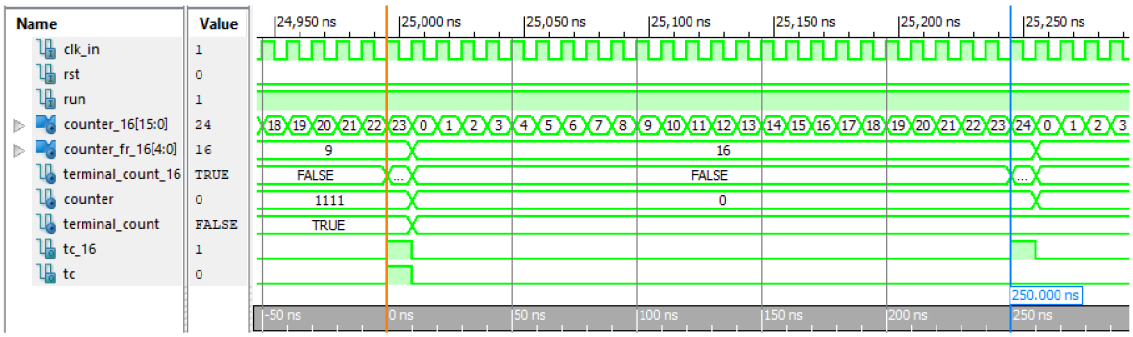
Simulace děličky frekvence přijímače

Na obrázku 3.11 je zobrazena simulace děličky frekvence použité pro přijímač. Přenosová rychlost je pro účely simulace nastavena stejně jako pro děličku vysílače na 256000 bit/s. Kurzorem je vyznačen okamžik spuštění děličky vstupním signálem *run*. Pokud signál run není aktivní, jsou všechny čítače vynulovány. V pravé části obrázku je vygenerovaný signál reset, který vynuluje čítače a děličku pozastaví.



Obr. 3.11: Simulace děličky frekvence přijímače, s vyznačeným okamžikem spuštění

Na obrázku 3.12 je přiblížený detail počítání čítačů mezi dvěma pulsy výstupního signálu *tc_16*. Hodnota čítače celé části dělitele *counter_16* je tedy rozdílná v případě, že hodnota čítače desetinné části dělitele *counter_fr_16* přesahuje rozsah nastavený generickým parameterem *DIV_FRACT_BITS*, určujícím velikost registru pro nastavení hodnoty desetinné části dělitele. V případě simulace je velikost tohoto registru výchozí, tedy 4 bity. Je tedy možné vidět, že interval čítače *counter_16* je v případě přenosu čítače *counter_fr_16* o jednu periody hodinového signálu kratší. Poměr delších a kratších intervalů koresponduje s velikostí desetinné části vypočítaného dělitele, dle rovnice 3.1 uvedené v kapitole 3.1.1.



Obr. 3.12: Detail výstupu simulace děličky frekvence přijímače se zobrazením počítání čítačů generujících výstupní signál *tc_16*.

3.2.4 Posuvný registr přijímače

Blok posuvného registru přijímače zajišťuje příjem sériových dat na vstupu *rx*. Entita obsahuje oproti posuvnému registru vysílače vyrovnávací buffer *rx_buffer*. Frekvence posuvu je řízena pouze vstupním signálem *clk_en*. Bity přijaté posuvným registrem *shift_reg* jsou přesunuty do bufferu, synchronně na náběžnou hranu vstupního hodinového signálu *clk_in* v případě, že je vstupní signál *ld_buf* na úrovni log. 1.

Vzorkování příchozích bitů

Příchozí data jsou vzorkována na šestnáctinásobné frekvenci, obdobně jako je popsané vzorkování dat SCI přijímačem mikrokontroléru M68HC1 v podkapitole 1.5.3. Pro určení pořadí vzorku příchozího bitu jsou pulsy vstupního signálu *clk_en_16* počítány čítačem *counter*, který je nulován vstupním signálem *clk_en* (začátek dalšího bitu). Uchovány jsou 3 vzorky vstupního signálu *rx* uprostřed příchozího bitu, tedy pokud je hodnota čítače 6 až 8. Vzorky jsou uloženy do interního 3-bitového registru *rx_samples*. Validní hodnota příchozího bitu je určena většinovou hodnotou vzorků a je uložena do interního signálu *rx_comp*, který je následně nasunut do posuvného registru a přiřazen na výstup *rx_cor*, reprezentující korektní hodnotu přijatého bitu. Pokud hodnoty všech vzorků nejsou stejné je indikováno zarušení přenosu a nastaven výstupní signál *noise_flag* na hodnotu log. 1.

Příjem paritního bitu

Volba využití a hodnoty paritního bitu je zvolena stejným způsobem jako u posuvného registru vysílače, jak je popsáno v podkapitole 3.1.2. Pro kontrolu příjmu paritního bitu je v procesu posuvného registru volána funkce *f_parity_check*, která je součástí knihovního balíku *parity.vhd*. Funkci je předáno nastavení druhu parity

par_opt a aktuální data v posuvném registru *shift_reg*. Návratovou hodnotou je informace, zda přijatý paritní bit souhlasí s nastavením. V okamžiku přesunu přijatých dat do bufferu je tato informace zapsána na výstup *par_chk* (v případě chybné hodnoty paritního bitu log. 1).

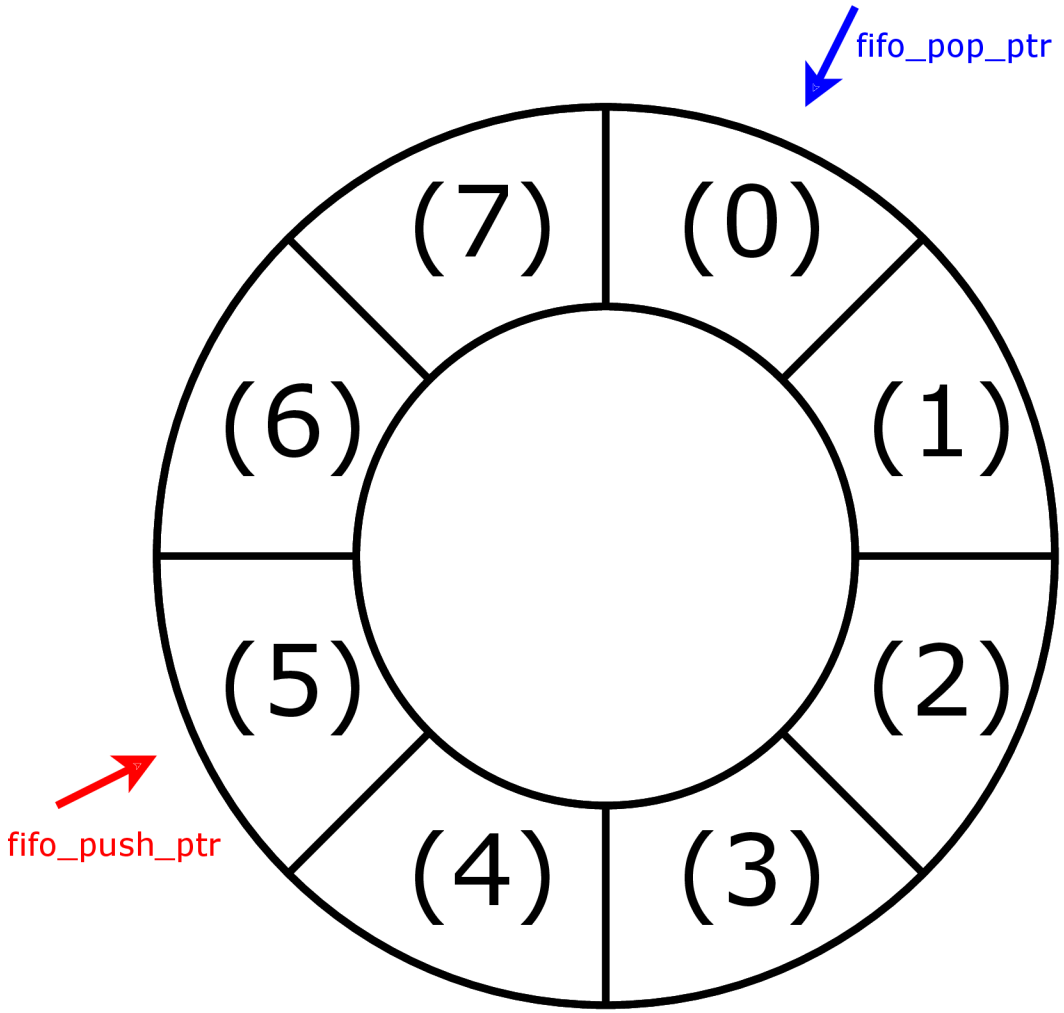
Implementace FIFO bufferu

Vyrovnávací buffer přijímače pracuje na principu kruhového pole, jak je popsáno v podkapitole 1.3. Ve VHDL popisu pro buffer deklarován typ *rx_fifo*, jako pole 8 bitových vektorů. Hloubka bufferu je parametrizovatelná pomocí generického parametru *FIFO_DEPTH*. Jako ukazatele zápisu a čtení na prvky pole jsou definovány signály *fifo_push_ptr* (FIFO Push Pointer) a *fifo_pop_ptr* (FIFO Pop Pointer), typu unsigned. Jelikož ukazatele jsou typu unsigned, musí být parametr *FIFO_DEPTH* vždy volen jako mocnina čísla 2, a přičemž je zajištěno správné fungování ukazatelů v kruhovém poli (přetečení rozsahu ukazatele navrací jeho hodnotu zpět na 0). K určení počtu bitů ukazatelů z hodnoty parametru *FIFO_DEPTH* je využito funkcí knihovního balíku `math_real`.

Na obrázku 3.13 je ilustrována indexace kruhového bufferu pro zvolenou hloubku bufferu 8 bajtů. Zápis do bufferu se provádí vždy s aktivním vstupním signálem *ld_buf* a při zápisu dojde k posunutí ukazatele *fifo_push_ptr* o jednu pozici vpřed. K naplnění bufferu dojde tehdy, pokud se ukazatel zápisu nachází o jednu pozici za ukazatelem čtení *fifo_push_ptr*. V tomto stavu nejsou přicházející data ukládána do bufferu a jsou ztracena. K přečtení dat z bufferu a posunu ukazatele čtení dochází při aktivním vstupním signálu *data_read*. K úplnému vyprázdnění bufferu dojde v případě, kdy jsou oba ukazatele na stejné pozici. V tomto případě není možné vstupem *data_read* posunout ukazatel čtení na pozici před ukazatel zápisu. Inspirací k realizaci FIFO paměti bylo řešení uvedené v [1] a princip kruhového bufferu uvedený v podkapitole 1.3 [17].

Realizace softwarového řízení datového toku

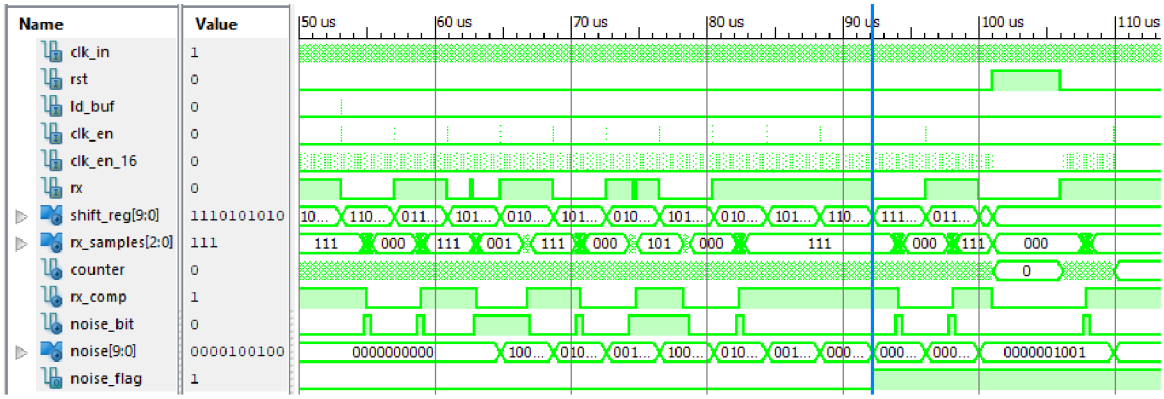
K indikaci zaplnění bufferu slouží výstupní signál *fifo_full*, který je aktivován v případě zaplnění bufferu z poloviny jeho hloubky. Míra zaplnění bufferu je zjištěna odečtením pozice ukazatele čtení od pozice ukazatele zápisu. Prázdná polovina bufferu by měla být dostatečnou rezervou pro příjem dat příchozích v době odesílání pozastavovacího znaku XOFF vysílačem. K deaktivaci výstupu *fifo_full* dojde po opětovném vyprázdnění bufferu na čtvrtinu jeho hloubky. Princip odesílání řídicích znaků vysílačem je popsán v podkapitolách 3.1.3 a 3.1.2.



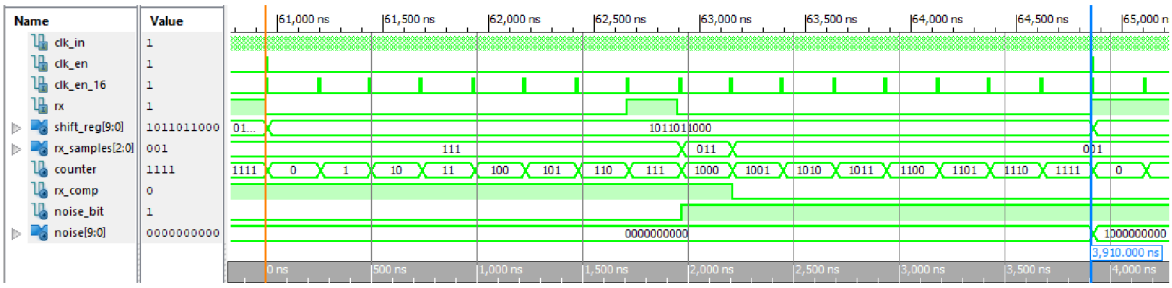
Obr. 3.13: Ilustrace FIFO bufferu přijímače o zvolené hloubce 8 bajtů

Simulace posuvného registru

Na obrázku 3.14 je zobrazena simulace funkce bloku posuvného registru při přijímání datového rámce s vzniklým rušením dvou datových bitů. Jak je možné vidět posuvný registr posouvá data při aktivní úrovni vstupu *clk_en*. Vzniklé rušení jednotlivých příchozích bitů signálu *rx* nastavuje interní signál *noise_bit*, jehož hodnota je nasunuta do registru *noise*. Kurzorem je označen okamžik přesunu dat do bufferu, kdy je nastaven výstup *noise_flag* (pokud na některém z bitů vzniklo rušení). V pravé části obrázku je vygenerovaný impuls vstupního signálu *rst*, který přednostně naplní vektor *shift_reg* hodnotami log. 1 a posuv může pokračovat s dalším impulsem vstupního signálu *clk_en*. Detail vzorkování jednoho bitu a vyhodnocení jeho hodnoty a možného rušení je na obrázku 3.15.

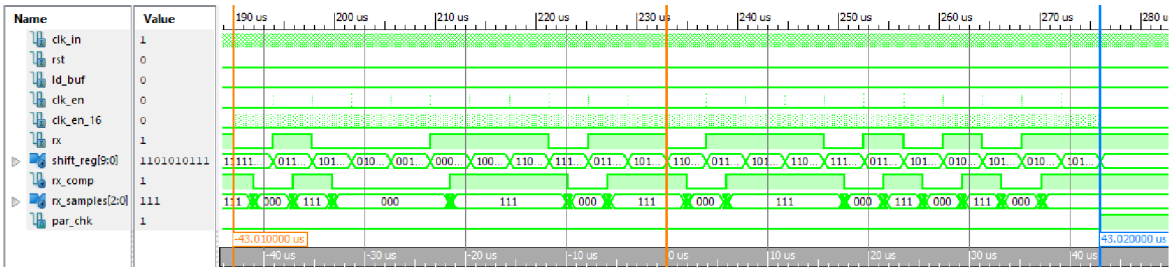


Obr. 3.14: Simulace posuvného registru přijímače s vygenerovaným zarušením dvou příchozích bitů a resetem



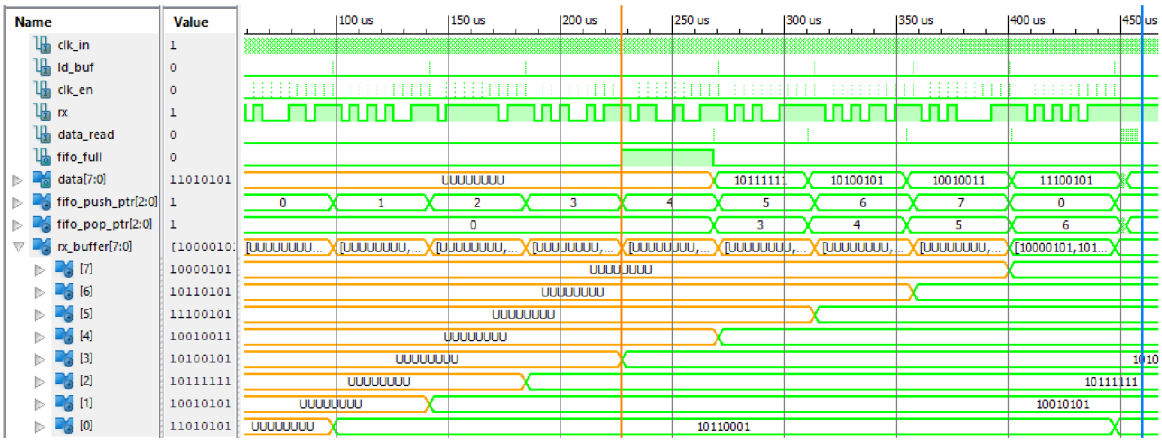
Obr. 3.15: Simulace detailního pohledu na vzorkování zarušeného bitu posuvným registrem přijímače

Simulace na obrázku 3.16 znázorňuje příjem dvou datových rámců s paritním bitem při nastavení liché parity. První rámec obsahuje paritní bit o správné hodnotě. Druhý rámec simuluje chybu paritního bitu, kdy je po příjmu této zprávy nastaven výstup *par_chk* na hodnotu log. 1.



Obr. 3.16: Simulace příjmu dvou datových rámců s paritním bitem, korektním i nekorektním

Pro simulaci FIFO bufferu, na obrázku 3.17, je nastavena hloubka bufferu 8 znaků. V okamžiku naplnění bufferu, označeném oranžovým kurzorem, je aktivován výstup *fifo_full*, dokud nejsou několika pulsy vstupního signálu *data_read* data přečtena. V pravé části obrázku je vidět, že ukazatel na pozici zápisu (*fifo_push_pos*) přeteče a inkrementuje se znovu od 0. V případě, že ukazatel na pozici čtení (*fifo_pop_pos*) dosáhne stejné pozice jako ukazatel zápisu, nelze ho dále posouvat (buffer je zcela vyprázdněn).

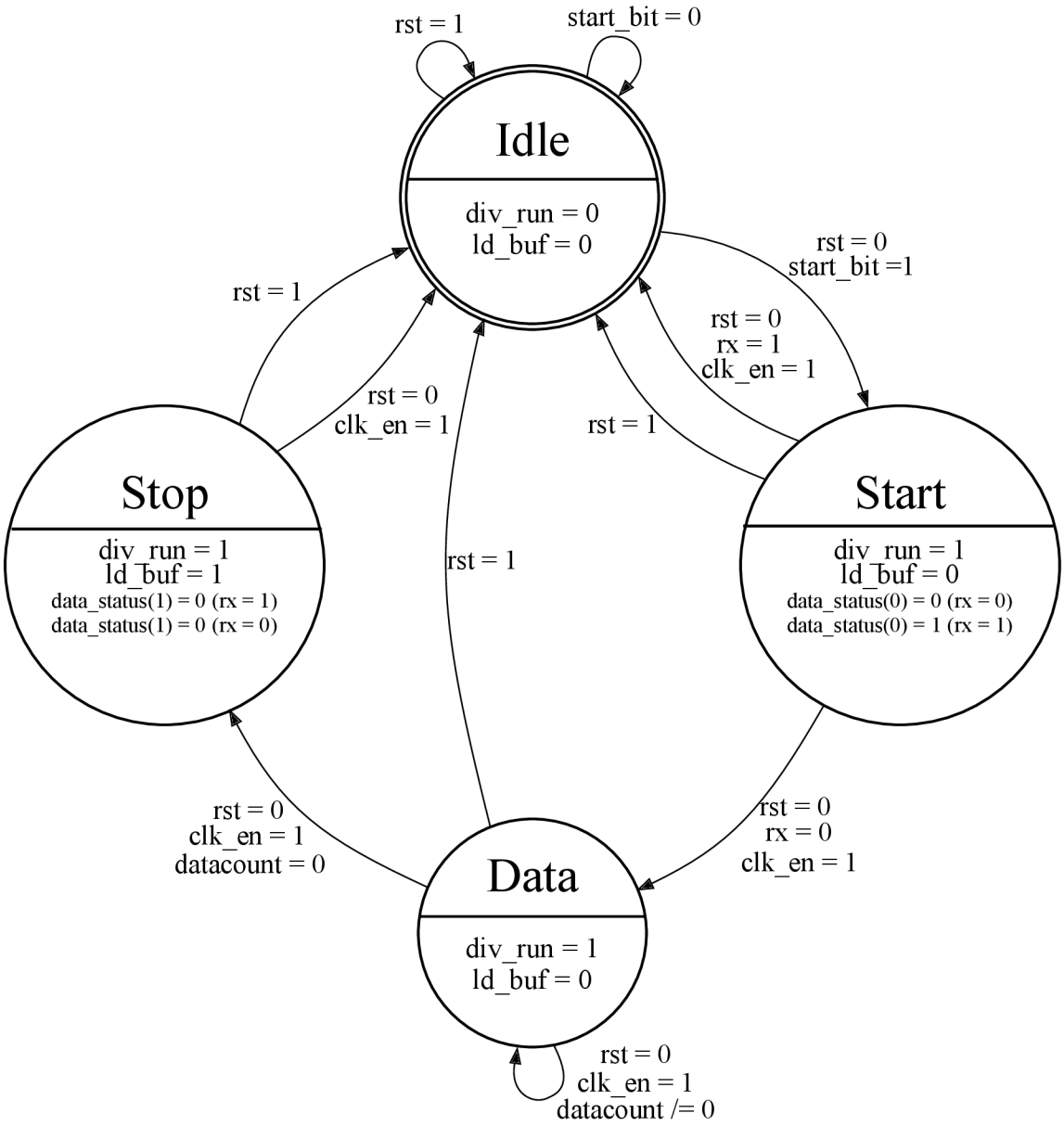


Obr. 3.17: Simulace funkce osmibajtového FIFO bufferu posuvného registru přijímače

3.2.5 Řídicí stavový automat přijímače

Hlavním řídicím blokem přijímače je konečný stavový automat Moorova typu. Stavový automat obsahuje stejně pojmenované stavy jako stavový automat vysílače a je popsán stavovým diagramem znázorněným na obrázku 3.18. Stavový automat je plně synchronní s *clk_in* a je aktivací vstupu *rst* navracen ze kteréhokoliv aktuálního stavu do stavu **Idle**. Ve stavu **Idle** čeká stavový automat na aktivaci vstupního signálu *start_bit*, což způsobuje přechod do stavu **Start**, čímž je aktivován výstupní signál *div_run*, který spouští chod děličky frekvence popsané v podkapitole 3.2.3. Přechod mezi ostatními stavy je také podmíněn aktivním vstupním signálem *clk_en*. Oproti stavovému automatu vysílače může návrat ze stavu **Start** do stavu **Idle** podmínit také neplatný vzorek start bitu při aktivním signálu *clk_en*. Platná hodnota příchozích dat je distribuována blokem posuvného registru, na vstup *rx*, jak je uvedeno v blokovém schématu na obrázku 3.8 (signál *rx_correct*). Touto podmínkou je také nastaven příslušný bit výstupního signálu *data_status* na hodnotu představující platný, či neplatný start bit.

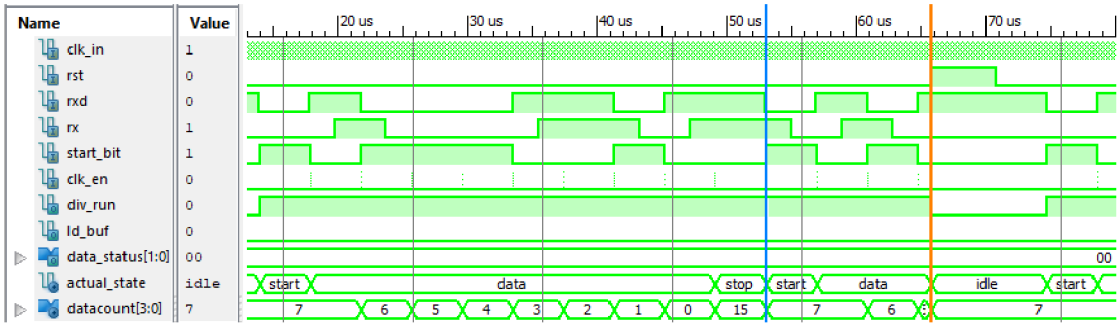
V případě platného start bitu přechází stavový automat do stavu **Data**, kde je důsledkem vstupního signálu *clk_en* dekrementován vnitřní signál *datacount*, indikující počet příchozích bitů nasunutých do posuvného registru popsaného v podkapitole 3.2.4. To znamená počet bitů, které ještě musí přijmout pro kompletní bajt. Pokud je *datacount* roven 0, přejde stavový automat do stavu **Stop** stejně, jako je tomu u stavového automatu vysílače. Ve stavu **Stop** je s následujícím pulsem vstupu *clk_en*, nastaven výstup *ld_buf* (Load Buffer) na hodnotu log. 1. Dále je nastaven přísušný bit výstupu *data_status* na hodnotu představující platný, či neplatný příjem stop bitu, podle aktuálního vzorku vstupního signálu *rx*.



Obr. 3.18: Stavový diagram řídicího stavového automatu přijímače

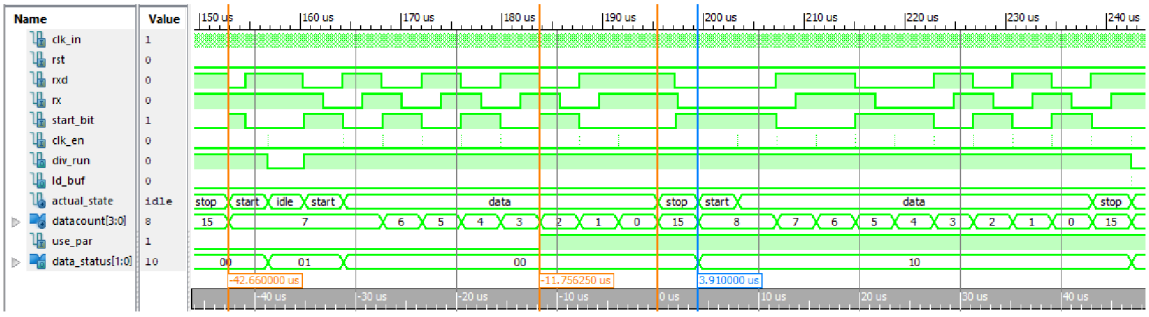
Simulace řídicího stavového automatu přijímače

Na obrázku 3.19 je zobrazena simulace základní funkce stavového automatu přijímače. Simulace obsahuje příjem korektního datového rámce a přerušení příjmu signálem *rst* v pravé části je nasimulován příjem datového rámce přerušený pulsem vstupního signálu. Okamžik resetu je na obrázku vyznačen oranžovou svislou čarou kurzoru. Výsledkem resetu je nastavení interního signálu *datacount* na počáteční hodnotu 7 a přechod automatu do stavu **Idle**.



Obr. 3.19: Simulace bloku řídicího stavového automatu přijímače, s vyznačeným okamžikem pulsu výstupu *ld_buf* a resetu

Simulace příjmu nekorektního datového rámce a zprávy s paritou je znázorněna na obrázku 3.20. Kurzory na obrázku při pohledu zleva zobrazují neplatný start bit, kdy je stavový automat navrácen do stavu **Idle** a je nastaven příslušný bit výstupu *data_status*. Další oranžový kurzor zvýrazňuje okamžik změny vstupního signálu *use_par*, čímž je zajištěna následující inicializace signálu *datacount* na hodnotu 8 (modrý kurzor). Poslední oranžový kurzor ukazuje na začátek neplatného stop bitu a je tak dále nastaven příslušný bit výstupu *data_status*.



Obr. 3.20: Simulace bloku řídicího stavového automatu přijímače s vygenerovanou chybnou hodnotou start bitu a stop bitu. Také s přepnutím příjmu paritního bitu signálem *use_par*.

3.2.6 Status registr přijímače

Výstupní registr *rx_data_status* signalizuje některou z forem porušení příchozích dat. Význam hodnot registru je uveden v tabulce 3.3. Hodnota může nabývat i kombinací jednotlivých možností.

Tab. 3.3: Význam bitů výstupního registru *rx_data_status* indikujícího platnost přijatého datového rámce

Hodnota *rx_data_status*	Význam
`0000` (0)	Příchozí datový rámec je platný
`0001` (1)	Start bit není platný
`0010` (2)	Stop bit není platný
`0100` (4)	Paritní bit není platný
`1000` (5)	Šum na některém z bitů rámce

3.3 Konfigurace UART

Po implementaci a zprovoznění návrhu UART je třeba nakonfigurovat parametry sériového přenosu dat. Parametry jsou přenosová rychlost, volba parity a využití softwarového řízení toku dat XON/XOFF.

3.3.1 Konfigurační registry

Top modul návrhu obsahuje 4 konfigurační registry. K nastavení přenosové rychlosti slouží registry *div_fract_reg*, *div_int_low_reg* a *div_int_high_reg*. Registr *div_fract_reg* slouží k nastavení zlomkové části dělitele hlavní hodinové frekvence a hodnota tohoto registru je dále distribuována na vstupy *div_fract* děliček frekvence. Šířka registru je volitelná volbou generického parametru *DIVIDER_FRACTION_BITS* v rozsahu 1 až 8 bitů. Registry *div_int_low_reg* a *div_int_high_reg* zprostředkují nastavení celé části dělitele. Způsob výpočtu dělitele je uveden v podkapitole 3.1.1. Spojení obou těchto registrů je přiřazeno na vstupy děliček frekvence *div_int*. Šířka spodního (low) registru je pevně určená na 8 bitů a šířka horního (high) registru je v rozsahu 1 až 8 bitů. Celková šířka je určená generickým parametrem *DIVIDER_INTEGER_BITS* v rozsahu 9 až 16 bitů. Poslední z registrů *config_reg* je 4 bitový, nastavuje paritu a zapíná použití SW řízení toku dat. Bity 0 a 1 nastavují vstupní signál *par_opt* posuvných registrů vysílače a přijímače. Bit 2 povoluje (log. 1) nebo zakazuje (log. 0) použití parity. Bit 3 povoluje (log. 1) nebo

zakazuje (log. 0) použití odesílání XON/XOFF znaků vysílače popsané v kapitole 3.1.

3.3.2 Nastavení adresy a hodnoty registru

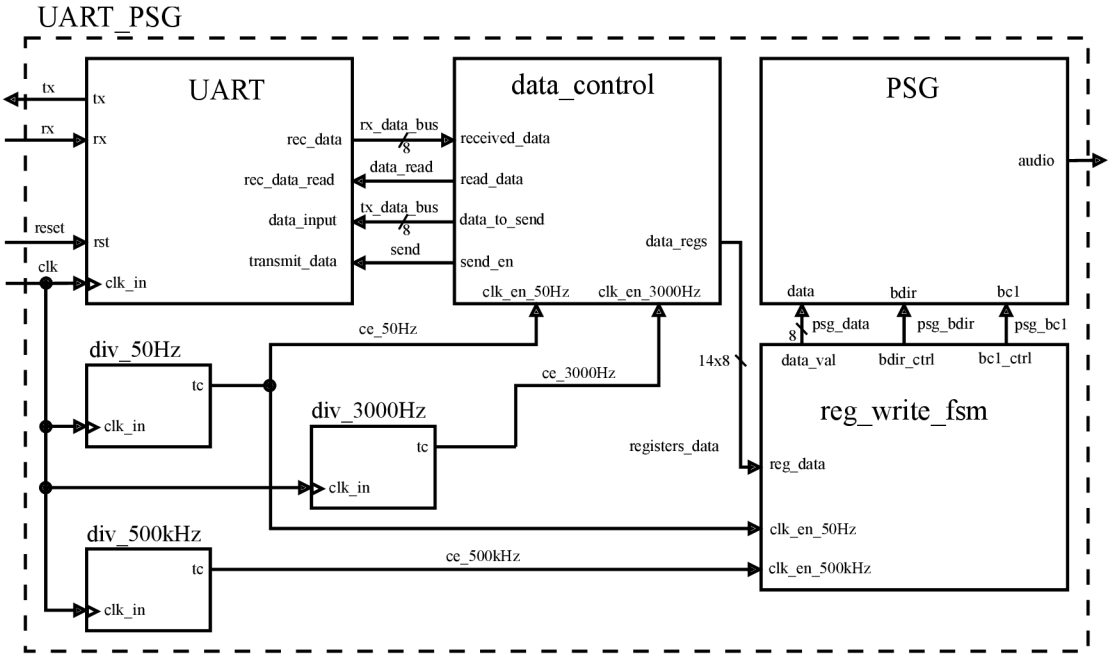
Hodnoty registrů a adresy jsou čteny z osmi přepínačů na vývojové desce NEXYS3. Adresa registrů je dvoubitová. Hodnota nastavená na přepínačích 0 a 1, je zapsána do registru adresy stiskem levého tlačítka (btnl). Hodnota registru je zapsána stiskem pravého tlačítka (btnr). Hodnoty adres jednotlivých registrů jsou uvedeny v tabulce 3.4. Zákmity stisků tlačítek jsou eliminovány debouncery navrženými v rámci úlohy č. 8 popsané v podkapitole 2.2.

Tab. 3.4: Hodnoty adres nastavovacích registrů

Adresa	Registr
00	config_reg
01	div_fract_reg
10	div_int_low_reg
11	div_int_high_reg

4 Realizace rozhraní pro přehrávání zvukových dat pomocí PSG

Pro přehrávání hudebních dat bylo využito navržené komunikační sběrnice UART popsané v kapitole 3. Na obrázku 4.1 je znázorněno blokové schéma navrženého řešení spojení UART a PSG. Tato kapitola se zabývá popisem komponent navržených pro správný přepis registrů PSG. Komunikační rychlost mezi PC a vývojovou deskou byla zvolena na 38400 bit/s. Hloubka bufferu přijímače byla nastavena na 64 bajtů.



Obr. 4.1: Blokové schéma navrženého rozhraní mezi UART a PSG (Přivedení synchronizačního signálu *clk* a resetu je pro některé komponenty zanedbáno)

4.1 Blok realizující příjem dat a zpracování kontrolního součtu

Entita pro zpracování přijatých dat je na blokovém schématu 4.1 označena jako *data_control* a je realizována konečným stavovým automatem. Přechody mezi stavy jsou řízeny vstupním povolovacím signálem *clk_en_3000Hz*. Výstupem *read_data* je ovládáno vyčítání z bufferu přijímače UART. Dle aktuálních dat na vstupu jsou postupně zkontrolovány hodnoty zpráv hlavičky odesílané programem v PC, který byl realizován v rámci bakalářské práce [3]. V případě správných hodnot těchto

zpráv jsou přijaté hodnoty registrů zapsány na výstup *data_regs*. Dalším krokem je ověřena hodnota kontrolního součtu přijatá v poslední zprávě. Podle správnosti kontrolního součtu je prostřednictvím UART odeslána potvrzovací zpráva. Formát zpráv byl navržen v bakalářské práci [3] a je popsán v podkapitolách 4.1.1 a 4.1.2.

4.1.1 Formát přijímaných zpráv z PC

. Zprávy přijímané vývojovou deskou obsahují hlavičku s hodnotami 81_{16}, 82_{16}, 83_{16}. Dále jsou přenesena zvuková data určená k přepisu registrů PSG. Kontrolní součet určující správnost přijatých dat je realizován exkluzivním logickým součtem hodnot všech registrů se zvukovými daty [3].

Tab. 4.1: Formát zpráv odesílaných programem v PC [3]

81_{16}	82_{16}	83_{16}	Registry $R0_{16}$ až RD_{16}	Kontrolní součet

4.1.2 Formát potvrzovacích zpráv odesílaných vývojovou deskou

. Hlavičku tvoří hodnota 83_{16}. Potvrzovací zpráva určuje zda mají být programem v PC odeslána následující data nebo aktuální data opakovaně. Pro opětovné zaslání dat je hodnota potvrzovací zprávy 41_{16} a 47_{16}. Jako požadavek o následující data jsou hodnoty 52_{16} a 45_{16}. Kontrolní součet je opět exkluzivní součet potvrzovacích zpráv [3].

Tab. 4.2: Formát potvrzovacích zpráv odesílaných vývojovou deskou [3]

83_{16}	Potvrzovací zpráva	Kontrolní součet

4.2 Blok realizující přepis registrů PSG

Pro změnu hodnot registrů PSG slouží blok *reg_write_fsm*. K přepisu registrů dochází s frekvencí 50 Hz určenou vstupním signálem *clk_en_50Hz*. Přepis registrů se provádí pomocí nastavení výstupního signálu *bdir_ctrl* na hodnotu log. 1 a přepínáním logických hodnot výstupu *bc1_ctrl*. Přepínání se provádí s každým pulsem vstupního signálu *clk_en_500kHz*. V případě *bc1_ctrl = 1* je zapsána adresa registru. V opačném případě je na výstup *data_val* zapsána hodnota daného registru. Hodnoty všech 14 registrů jsou přivedeny na vstup *reg_data*.

5 Analýza návrhu pomocí ChipScope

V návrhovém prostředí Xilinx ISE je podporováno použití nástroje **ChipScope Pro** pro testování návrhu uvnitř FPGA a měření jeho parametrů. Spojení testovaného návrhu s PC je zajištěno pomocí JTAG (Joint Test Action Group) portu FPGA obvodu Spartan-6. Pro zajištění dostatečně rychlého přenosu naměřených dat do PC je obvod Spartan-6 vybaven 8 GTP sériovými vysílači/přijímači, které mohou odesílat naměřená data do PC přenosovou rychlostí až 3,2 Gb/s [9][10].

5.1 Jednotlivé prvky pro testování pomocí ChipScope

Použití nástroje ChipScope vyžaduje implementaci následujících součástí do FPGA.

5.1.1 ILA core testovací součást

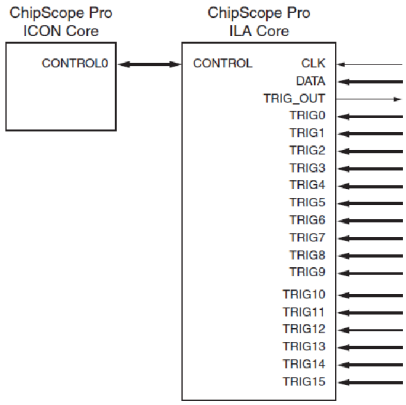
ILA (Integrated Logic Analyzer) core je součást implementovaná do hradlového pole pro měření signálů návrhu pomocí ChipScope Pro. Na definované vstupy ILA core lze přiřadit libovolné signály uživatelského návrhu. Vstupy ILA jsou spouštěcí (trigger) signály, datové signály a signál hodinové frekvence, ke které jsou následně vztaženy zachycené vzorky datových signálů. Jako vstup hodinového signálu je dobré volit hlavní hodinovou frekvenci celého návrhu. Podmínky pro spuštění měření dle signálů přivedených na *TRIG* vstupy mohou nabývat všech relací (=, <>, >, <, >=, <=) [12].

5.1.2 ICON core testovací součást

ICON zajišťuje rozhraní mezi implementovaným ILA a JTAG portem hradlového pole. Jedna součást ICON může poskytnout spojení až 15 ILA komponent. Na obrázku 5.1 je principiální blokové schéma propojení součástí ICON a ILA [11][12].

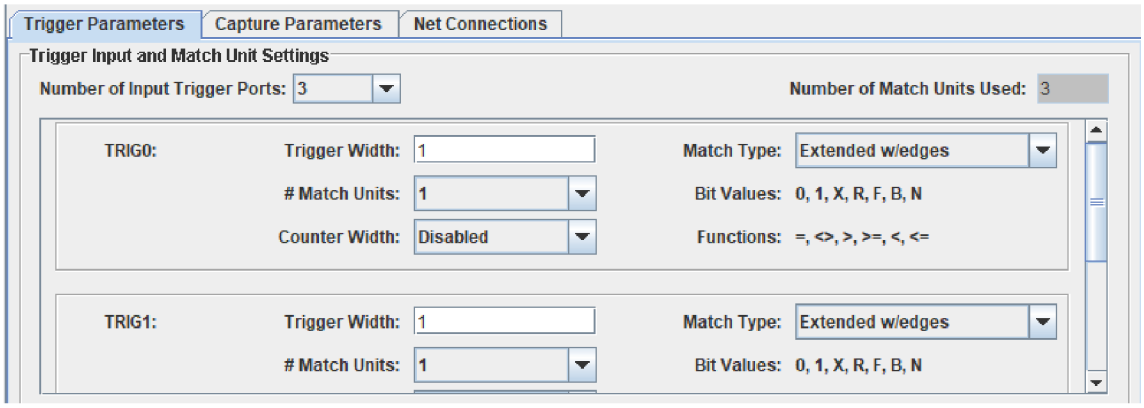
5.2 Konfigurace Chipscope

Použití nástroje ChipScope vyžaduje přidání konfiguračního souboru s příponou *.cdc* (ChipScope Definition and Connection File). Konfigurace tohoto souboru umožňuje vložit do návrhu ICON a ILA součásti. Konfigurace se provádí pomocí programu Core Inserter. Při nastavení Nastavení ILA je nejprve zapotřebí zvolit počet spouštěcích vstupů TRIG0-15. Pro každý z nich je pak možné volit jejich šířku a možnosti zachytávání jejich změn a logických úrovní, jak je vidět na obrázku 5.2. Na obrázku 5.3 je zobrazeno nastavení zachytávání dat. Vzorkování je možné volit na náběžnou

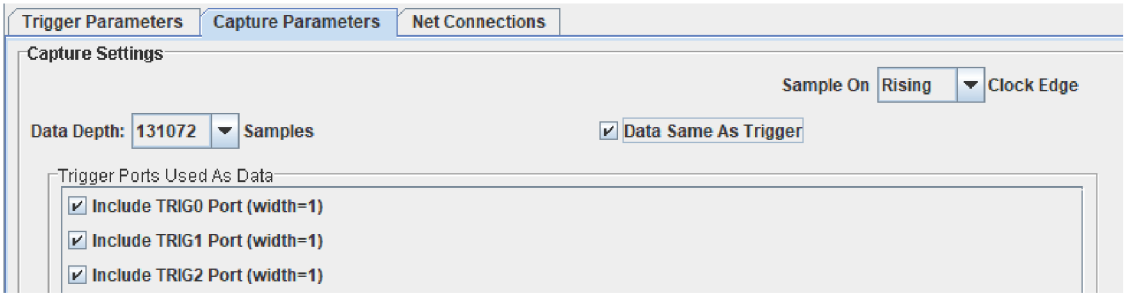


Obr. 5.1: Spojení součástí ICON a ILA sloužící pro testování návrhu pomocí nástroje Chipscope Pro [12]

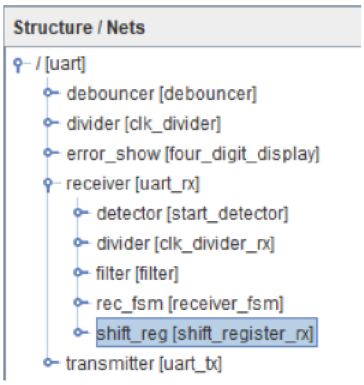
nebo sestupnou hranu hodinového signálu. Měřené signály mohou být stejné jako spouštěcí, což je výhodnější z důvodu menšího využití zdrojů FPGA. Další volbou je možné nastavit počet zachytávaných vzorků. Kombinace maximálního počtu spouštěcích, datových signálů a hloubky vzorkovacího bufferu je omezen počtem BRAM (block RAM). Připojení požadovaných signálů je možné v záložce Net Connections. Pro zobrazení celé hierarchie návrhu, jak je vidět na obrázku 5.4, je třeba v nastavení syntézy nastavit možnost *keep hierarchy* na *soft*. Jako hodinový signál pro měření je třeba připojit hlavní hodinovou frekvenci viz obrázek 5.5. V záložce Trigger/Data Signals lze připojit spouštěcí signály, což je zobrazeno na obrázku 5.6. V případě, že požadovaná data pro sledování nejsou stejná jako spouštěcí signály, je výběr separovaný.



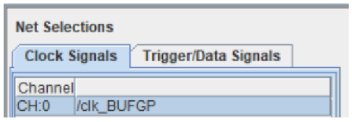
Obr. 5.2: Nastavení ILA core spouštěcích parametrů



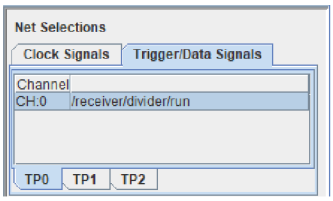
Obr. 5.3: Nastavení ILA core spouštěcích parametrů



Obr. 5.4: Možnost výběru signálů pro ILA z celého návrhu.



Obr. 5.5: ILA core - připojení signálu *clk*.



Obr. 5.6: ILA core - připojení signálu pro spouštění měření.

5.3 Zobrazení měřených průběhů pomocí ChipScope

Po nastavení všech parametrů pro testování návrhu a implementaci návrhu do FPGA je možné sledovat průběhy sledovaných signálů. Ke sledování měřených hodnot slouží program ChipScope Pro Analyzer, který má možnost nastavit úrovně spouštění měření podle hodnot jednotlivých signálů. V podokně waveform je možné sledovat průběhy sledovaných signálů.

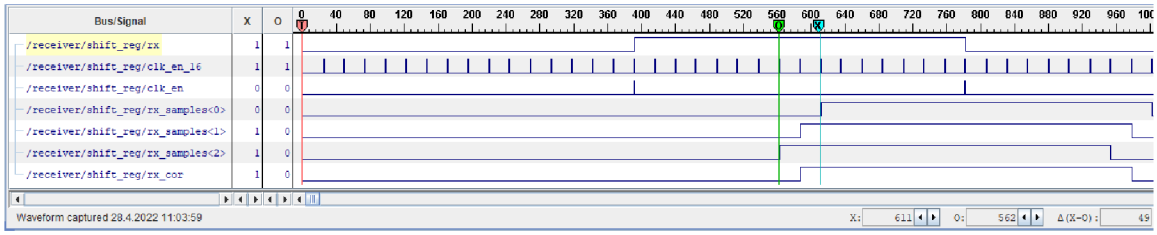
5.4 Příklady časových průběhů naměřených pomocí ChipScope

Nástroj ChipScope Pro byl využit k ověření nastavení přenosové rychlosti UART. Na obrázku 5.7 je zobrazený průběh výstupů *tc_16* a *tc* děličky frekvence přijímače pro nastavenou přenosovou rychlost 256000 bit/s. Na tuto přenosovou rychlost připadá perioda jednoho bitu $bit\_time = 1/256000 = 3,91\ \mu s$, čemuž odpovídá 391 vzorků

podle hodinové frekvence 100 MHz signálu *clk*. Z průběhu na obrázku je patrné, že přenosová rychlost odpovídá požadavkům. Tato skutečnost byla ověřena na několika přijatých bitech.

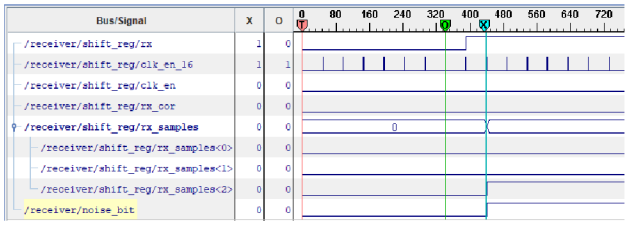
Obr. 5.7: Průběh analýzy výstupů děličky frekvence přijímače nastavené na přenosovou rychlost 256000 bit/s.

Dalším příkladem ověření funkčnosti návrhu je vzorkování příchozích bitů posuvným registrem přijímače. Tato funkce je popsána v podkapitole 3.2.4. Na obrázku 5.8 je uvede průběh signálů při vzorkování bitu. Kurzory O a X je označena oblast vzorkování třech vzorků a na posledním řádku je vidět změna průběhu výstupního signálu *rx_cor*, která je určena většinovou hodnotou registru *rx_samples*.



Obr. 5.8: Průběh analýzy vzorkování příchozích bitů přijímačem

Na obrázku 5.9 je naměřený vznik rušení, které je vygenerováno pomocí nastavení odlišné přenosové rychlosti. Kurzory jsou opět označeny tři vzorky, z nichž dva mají hodnotu log. 0 a třetí log. 1. V důsledku toho je nastaven signál *noise_bit* zobrazený na posledním řádku.



Obr. 5.9: Průběh analýzy vzorkování příchozích bitů přijímačem s vygenerovaným rušením

6 Návrh laboratorní úlohy

Laboratorní úloha slouží jako demonstrace fungování UART a je rozdělena do dvou úloh, které jsou uvedeny v podkapitolách 6.3 a 6.4. V druhé laboratorní úloze je připraveno blokové schéma, které zajistí spojení navrženého vysílače a přijímače s PC, viz kapitola 6.4. Po realizaci tohoto spojení bude možné využít návrh k přehrávání hudby pomocí navrženého zvukového generátoru PSG [3].

6.1 Vývojová deska NEXYS3

Při realizaci laboratorní úlohy bude použita vývojová deska NEXYS3 od firmy DIGILENT. Vývojová deska je osazena hradlovým polem Spartan-6 LX16 od výrobce Xilinx. Pro sériovou komunikaci je na vývojové desce osazen převodník USB-UART FTDI FT232. Pro řízení toku dat je využíváno softwarové řízení vysvětlené v podkapitole 1.4.2 [8].

Deska je osazena CMOS oscilátorem o frekvenci 100 MHz, který bude sloužit jako hlavní hodinový signál *clk* pro UART navržený v kapitole 3 [8].

6.2 Převodník FTDI FR232

Převodník FR232 podporuje verze USB 1.1/2.0. Komunikační rychlosti převodníku lze volit ze všech standardních nebo jsou libovolně volitelné v rozsahu 183 bit/s až 3 Mbit/s. Pro odesílání dat přes USB sběrnici a příjem dat z UART, je v převodníku obsažen TX Buffer o velikosti 256 bytů (TX vzhledem k USB rozhraní). V případě příjmu dat z USB rozhraní má RX Buffer velikost 128 bytů. Buffery jsou paměti typu FIFO a data z nich jsou zapisována/čtena UART vysílacím/přijímacím registrem. Převodník může zajistit také konverzi na rozhraní RS232, RS422 nebo RS485. Převodník umožňuje hardwarové i softwarové řízení toku dat, které je podrobněji vysvětleno v kapitole 1.4 [7].

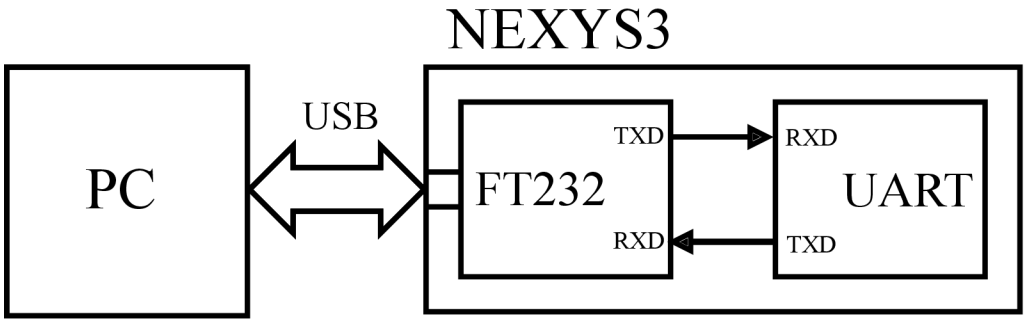
6.3 Laboratorní úloha návrhu programovatelné děličky frekvence

Zadání první laboratorní úlohy je dostupné v příloze A. Studentům bude v této laboratorní úloze dostupný projekt *UART_VHDL*. Studenti budou mít za úkol doplnit chybějící architektury entit děliček frekvence vysílače i přijímače. Dalším úkolem bude vytvořit entitu *debouncer* sloužící pro ošetření zákmitů tlačítek. Kompletací a

testováním návrhu UART se zabývá druhá laboratorní úloha uvedená v podkapitole 6.4.

6.4 Laboratorní úloha pro spojení s PC

Pro komunikaci navrženého vysílače/přijímače s PC je navrženo blokové uspořádání dle obrázku 6.1. Toto zapojení bude sloužit k odesílání a příjmu dat mezi vývojovou deskou a PC. Pro demonstraci odesílání a příjmu dat může být v PC využitý klient PuTTy, který může být nakonfigurován také na sériovou komunikaci.



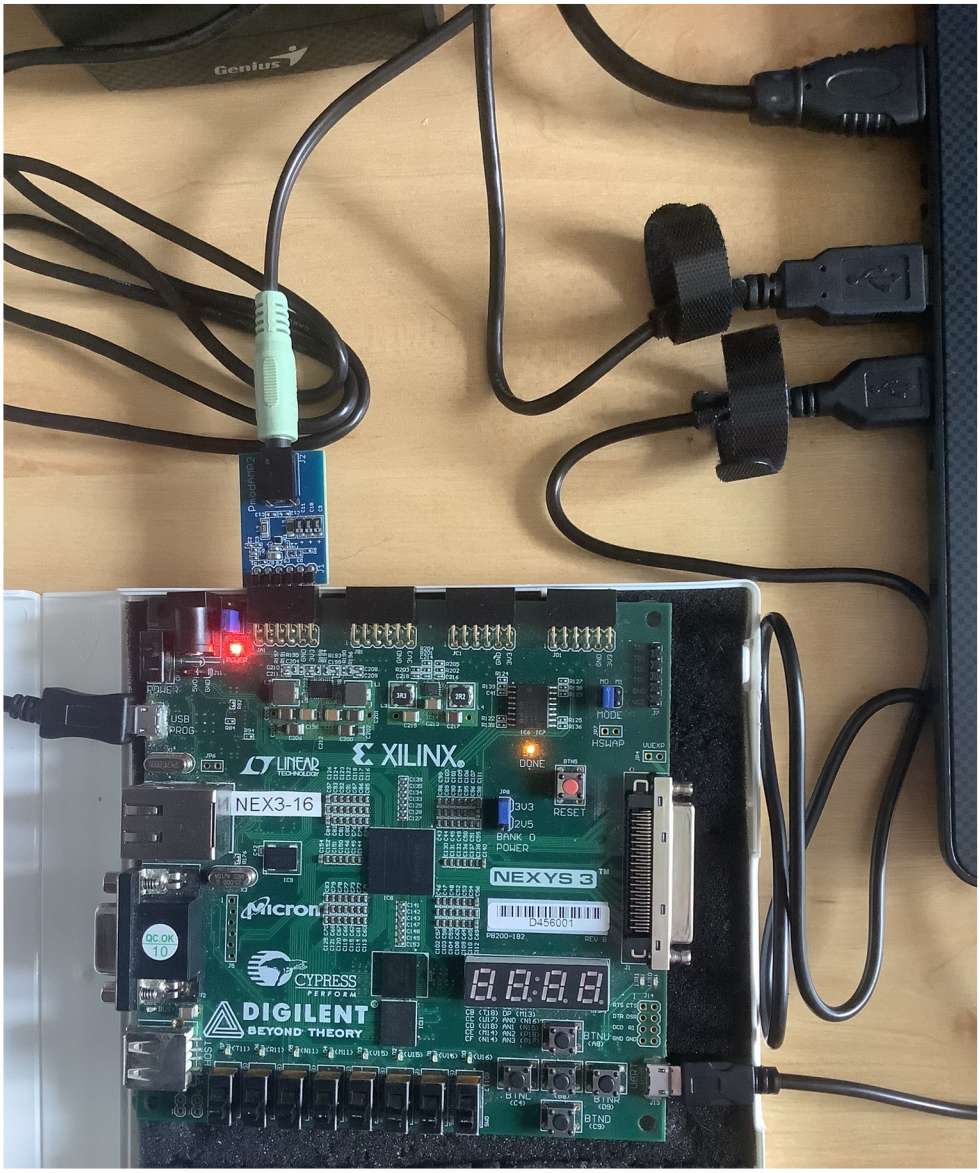
Obr. 6.1: Blokové uspořádání laboratorní úlohy realizující spojení navrženého vysílače/přijímače s PC

6.4.1 Využití periferií vývojové desky k demonstraci funkce UART

Hodnota odesílaných znaků vývojovou deskou do PC lze nastavit pomocí osmi přepínačů na desce. Hodnota příchozího znaku je zobrazena pomocí dvou pravých segmentů sedmi-segmentového displeje. Zobrazení příchozí hodnoty je reprezentováno jeho hexadecimální hodnotou podle ASCII tabulky. Nastavení parametrů sériového přenosu je blíže popsáno v podkapitole 3.3.1. Vstupní signál *tx_send* pro odeslání nastavených dat je připojen na prostřední tlačítko *btns*. Pro možnost synchronního resetu bylo zvoleno spodní tlačítko *btnd*. Zákmity vzniklé při stisku těchto tlačítek jsou ošetřeny debouncery. Horní tlačítko *btnu* je určeno pro vyčtení přijatých dat z bufferu, tedy vstup *rx_data_read*. Jelikož data jsou z bufferu vyčítána s každou náběžnou hranou hlavní hodinové frekvence *clk* (za předpokladu aktivního signálu *rx_data_read*), je využitý výstup *ce* debounceru pro vygenerování pulsu s délkou jedné periody hodinové frekvence.

6.4.2 Zadání laboratorní úlohy

Zadání druhé laboratorní úlohy je dostupné v příloze B. Prvním úkolem je doplnit architekturu entity řídicího stavového automatu přijímače v projektu *UART_VHDL*. Dalším úkolem je otestovat funkčnost celého návrhu UART, zapojeného podle blokového schématu na obrázku 6.1. Způsob zobrazování a odesílání dat pomocí periferií vývojové desky je uveden v podkapitole 6.4.1. Poslední úkolem je využít navržené entity k přehrávání hudebních dat pomocí PSG. Studentům bude poskytnutý projekt *UART_PSG*, který umožňuje přehrávat hudební data. Na obrázku 6.2 je fotografie kompletního zapojení vývojové desky pro přehrávání hudebních dat.



Obr. 6.2: Fotografie kompletního zapojení vývojové desky pro přehrávání zvukových dat, jehož návrh je uvedený v kapitole 4

Závěr

Cílem práce bylo navrhnout asynchronní sériový vysílač/přijímač a implementovat jej do hradlového pole. Dále byl cílem také návrh propojení asynchronního sériového vysílače/přijímače s PC pro demonstraci jeho funkce v rámci laboratorní úlohy kurzu „Logické obvody a systémy“, což je popsáno v kapitole 6.

Popis fungování UART v první kapitole byl základem výsledného návrhu popsaného v kapitole 3. Pro nastudování funkce UART byla při tvorbě návrhu nastudována již existující architektura UART (SCI) mikrokontroléru M68HC11, jejíž funkcionalita je krátce přiblížena v kapitole 1.5. Jako inspiraci pro popis UART v jazyce VHDL jsem použil existující implementaci uvedenou v [2]. Oproti této implementaci je výsledný návrh v této práci rozdělen na dílčí entity a vhodně sestaven pomocí strukturálního popisu, jak je vidět na blokových schématech vysílače a přijímače, uvedených na obrázcích 3.1 a 3.8. Tato implementace je bližší laboratorním úlohám kurzu „Logické obvody a systémy“ a obecně způsobu jeho výuky.

Návrh UART umožňuje změnu přenosové rychlosti pomocí tlačítek a přepínačů na vývojové desce NEXYS 3. Pro generování frekvence přenosové rychlosti byla implementována programovatelná dělička frekvence s možností dělení hlavní hodinové frekvence, dělitelem s uvážením desetinné části jeho hodnoty. Popis výpočtu dělitele je uveden v podkapitole 3.1.1. Pro detekci rušení sériové linky a stanovení korektní hodnoty přijímaných bitů bylo implementováno vzorkování každého bitu na šestnáctinásobné frekvenci. Platná hodnota každého z bitů je určena většinovou logickou hodnotou 3 vzorků daného bitu. Tato funkce byla inspirována SCI modulem mikrokontroléru M68HC11, jak je popsáno v podkapitole 1.5.3. Návrh je také schopen odesílat a přijímat paritní bit standardně používaných hodnot.

Pro realizaci možnost přijímání většího počtu zpráv z PC, byla do přijímače implementována vyrovnávací paměť typu FIFO. Velikost tohoto bufferu je možné parametrizovat při syntéze návrhu. Detailní popis principu fungování navržené vyrovnávací paměti je uveden v podkapitole 3.2.4. Pro případ výrazně pomalejšího čtení přijatých dat než jejich odesílání protistranou byla implementována funkcionalita softwarového řízení toku dat. Způsob implementace odesílání příslušných znaků pozastavujících a obnovujících přenos je blíže popsán v podkapitole 3.1.2.

Kapitola 4 popisuje navržené řešení umožňující přehrávání zvukových dat, odesílaných programem v PC, pomocí realizovaného zvukového generátoru. Rozhraní mezi UART a PSG je nakonfigurováno na pevnou přenosovou rychlost a frekvenci přepisu registrů zvukového generátoru.

Pro navržený UART byla vytvořena dvě zadání studenských laboratorních úloh, které jsou obsahem příloh A a B. V prvním zadání budou mít studenti za úkol implementovat programovatelnou děličku frekvence. Druhé zadání vyžaduje navrhnout

stavový automat UART přijímače a dokončit tak celý UART. V případě, že studenti dokončí zadané úlohy mohou si vyzkoušet odesílat a přijímat data mezi vývojovou deskou a PC. Posledním úkolem je využít zhotovený UART k přenosu zvukových dat pro zvukový generátor. Připravené projekty pro realizaci studentských úloh jsou obsahem elektronické přílohy této práce. Kompletní verze byla předána vedoucímu práce, tak aby řešení úloh nebylo studentům dostupné na internetu.

Realizovaný UART je zcela funkční. Hlavní výhodou návrhu je možnost nastavování přenosové rychlosti na širokou škálu přenosových rychlostí a možnost pozastavování přenosu pomocí softwarového řízení toku dat, které je ovšem implementováno pouze ve směru příjmu. Pro demonstraci jeho funkce bylo nahráno video, které je součástí elektronické přílohy. Přehrávání zvukových dat je demonstrováno nahrávkou přehrání dodaných skladeb.

Navržený UART je možné dále doplnit o vyrovnávací paměť vysílače a realizovat pozastavování odesílání dat v případě požadavku protistrany. Po detekci start bitu by mohla být jeho hodnota ověřena pomocí více vzorků šestnáctinásobné vzorkovací frekvence a podle jejich hodnot detekovat případné poškození start bitu. Dalším možným vylepšením může být doplnění výstupního signálu, který bude poskytovat informaci o přijetí nových dat. Pro přijímač je také možné implementovat uchování informací o správnosti přijatého bitu a o rušení, pro všechny přijaté hodnoty uložené v bufferu. V případě využití návrhu na jiné vývojové desce s možností připojení signálů pro hardwarové řízení toku dat by bylo vhodné jej do návrhu doplnit.

Rozhraní pro spojení UART se zvukovým generátorem popsané v kapitole 4 je možné vylepšit o možnost změny frekvence přepisu registrů PSG. Výpočet kontrolního součtu by bylo možné realizovat přímo v rámci UART komponenty či jiným sofistikovanějším způsobem, aby nebylo zapotřebí ukládat přijatá data do zvláštního pole registrů.

Práce může být dále rozšířena o využití UART k jinému účelu než je přehrávání zvukových dat. Příkladem může být příjem hodnot pro další zpracování pomocí pulsně šířkové modulace, například pro řízení motorů či digitálně-analogový převod pro LED diodu. Navržený UART může být také využit jako komunikační rozhraní pro mikroprocesor. Dalším využitím může být odesílání hodnot z různých snímačů do PC.

Realizace UART byla časově náročná a odhadem jsem samotnému návrhu věnoval minimálně 250 hodin. Návrh a realizace rozhraní pro přehrávání zvukových dat pomocí PSG bylo věnováno přibližně 50 hodin a návrhu zadání studentských úloh bylo věnováno obdobné množství času.

Realizovaný UART byl také prezentován jako soutěžní příspěvek v rámci studentské konference EEICT.

Literatura

[1] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. Praha: BEN - technická literatura, 2006. ISBN 80-730-0198-5.

[2] MALÝ, Martin. *Data, čipy, procesory: vlastní integrované obvody na koleni*. Praha: CZ.NIC, z.s.p.o., 2020. CZ.NIC. ISBN 978-80-88168-53-9.

[3] HOMZOVÁ, Eliška. *Realizace výukového vícekanálového zvukového obvodu* [online]. Brno, 2021 [cit. 2021-12-13]. Dostupné z: http://hdl.handle.net/11012/198025. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky. Vedoucí práce Petr Petyovský.

[4] PETYOVSKÝ, Petr. *Předmět Logické obvody a systémy: Úloha č. 6: Sekvenční logika — návrh asynchronních a synchronních binárních čítačů, výhody a nevýhody, využití*. Rev. 4. Brno.

[5] PETYOVSKÝ, Petr. *Předmět Logické obvody a systémy: Úloha č. 8: Sekvenční logické obvody - konečný stavový automat, debouncer, měření doby stisknutí tlačítka, čítače*. Rev. 3. Brno.

[6] *M68HC11 Reference Manual* [online]. 2007 [cit. 2021-11-28]. Dostupné z: https://www.nxp.com/docs/en/reference-manual/M68HC11RM.pdf

[7] *FTDI FT232R: Datasheet* [online]. [cit. 2021-12-22]. Dostupné z: https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf

[8] *Nexys3 Reference Manual* [online]. 1300 Henley Court | Pullman: Digilent, 2013 [cit. 2021-12-22]. Dostupné z: https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys3/documentation/Nexys3_rm.pdf

[9] *Spartan-6 Family Overview* [online]. 2100 Logic Drive San Jose, CA 95124: Xilinx, 2011 [cit. 2022-04-27]. Dostupné z: https://docs.xilinx.com/v/u/en-US/ds160

[10] *ChipScope Pro* [online]. 2100 Logic Drive San Jose, CA 95124: Xilinx, 2012 [cit. 2022-04-06]. Dostupné z: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx14_7/chipscope_pro_sw_cores_ug029.pdf

[11] *ChipScope Pro ICON* [online]. 2100 Logic Drive San Jose, CA 95124: Xilinx, 2011 [cit. 2022-04-27]. Dostupné z: https://docs.xilinx.com/v/u/en-US/chipscope_icon

[12] *ChipScope Pro ILA* [online]. 2100 Logic Drive San Jose, CA 95124: Xilinx, 2011 [cit. 2022-04-27]. Dostupné z: https://docs.xilinx.com/v/u/en-US/chipscope_ila

[13] *SCC2691: Datasheet* [online]. 2006 [cit. 2021-12-23]. Dostupné z: https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf

[14] *MAX3108: Datasheet* [online]. 160 Rio Robles, San Jose, CA 95134 USA: Maxim Integrated, 2013 [cit. 2022-03-24]. Dostupné z: https://datasheets.maximintegrated.com/en/ds/MAX3108.pdf

[15] Universal asynchronous receiver-transmitter. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-11-29]. Dostupné z: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

[16] Asynchronous serial communication. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-11-28]. Dostupné z: https://en.wikipedia.org/wiki/Asynchronous_serial_communication

[17] Circular buffer. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-12]. Dostupné z: https://en.wikipedia.org/wiki/Circular_buffer

[18] Flow control (data). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 8. 6. 2021 [cit. 2021-12-23]. Dostupné z: https://en.wikipedia.org/wiki/Flow_control_(data)

[19] Software flow control. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 4. 1. 2020 [cit. 2021-12-23]. Dostupné z: https://en.wikipedia.org/wiki/Software_flow_control

[20] RS-232. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 11. 12. 2021 [cit. 2021-12-27]. Dostupné z: https://en.wikipedia.org/wiki/RS-232

[21] Serial port. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Serial_port

[22] Data terminal equipment. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2.6.2021 [cit. 2021-12-27]. Dostupné z: https://en.wikipedia.org/wiki/Data_terminal_equipment

[23] Data circuit-terminating equipment. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 30. 4. 2021 [cit. 2021-12-27]. Dostupné z: https://en.wikipedia.org/wiki/Data_circuit-terminating_equipment

Seznam symbolů a zkratek

UART	Asynchronní sériový vysílač/přijímač
SCI	Serial Communication Interface
LSB	Nejméně významný bit
MSB	Nejvíce významný bit
PSG	Programovatelný zvukový generátor
LFSR	Lineární zpětnovazební čítač
VHDL	VHSIC Hardware Description Language
FPGA	Programovatelné hradlové pole
FIFO	First In, First Out
RTS	Request To Send
CTS	Clear To Send
DTR	Data Terminal Ready
DSR	Data Set Ready
DTE	Data Terminal Equipment
DCE	Data Circuit-terminating Equipment
JTAG	Joint Test Action Group
ILA	Integrated Logic Analyzer
ICON	Integraterd CONtroller

Seznam příloh

A Zadání studentské laboratorní úlohy pro návrh programovatelné děličky frekvence UART **58**

B Zadání studentské laboratorní úlohy pro dokončení a zprovoznění UART **61**

C Obsah elektronické přílohy **64**

A Zadání studentské laboratorní úlohy pro návrh programovatelné děličky frekvence UART

Cíle

- Návrh programovatelné děličky frekvence vysílače i přijímače UART pomocí synchronních čítačů a principu sigma-delta modulace.
- Návrh debounceru pomocí konečného stavového automatu.

Teoretický úvod

Čítače

Čítač je zařízení určené k počítání impulsů vstupního hodinového signálu. Čítače mohou být synchronní nebo asynchronní vůči vstupnímu hodinovému signálu. Asynchronní čítače mají značnou nevýhodu spočívající ve velkém zpoždění. U těchto čítačů je vstupní hodinový signál přiveden pouze na první klopný obvod, ostatní klopné obvody jsou pak zapojeny v kaskádě a vzniká tak na jejich vstupech zpoždění vstupního signálu. Použití synchronního čítače tuto nevýhodu eliminuje, vstupní hodinový signál je přiveden na všechny klopné obvody čítače a ty tak pracují paralelně. Malou nevýhodou synchronního čítače je potřebnost výpočtu následujícího stavu čítače pomocí kombinační logiky. Čítače mohou být dále rozděleny dle kódu, ve kterém počítají, na binární, BCD, Grayův kód a modulo M čítač. Právě čítač modulo M lze využít k implementace děličky frekvence. Dále mohou být čítače rozděleny dle směru počítání na dopředné, zpětné a zpětnovazební.

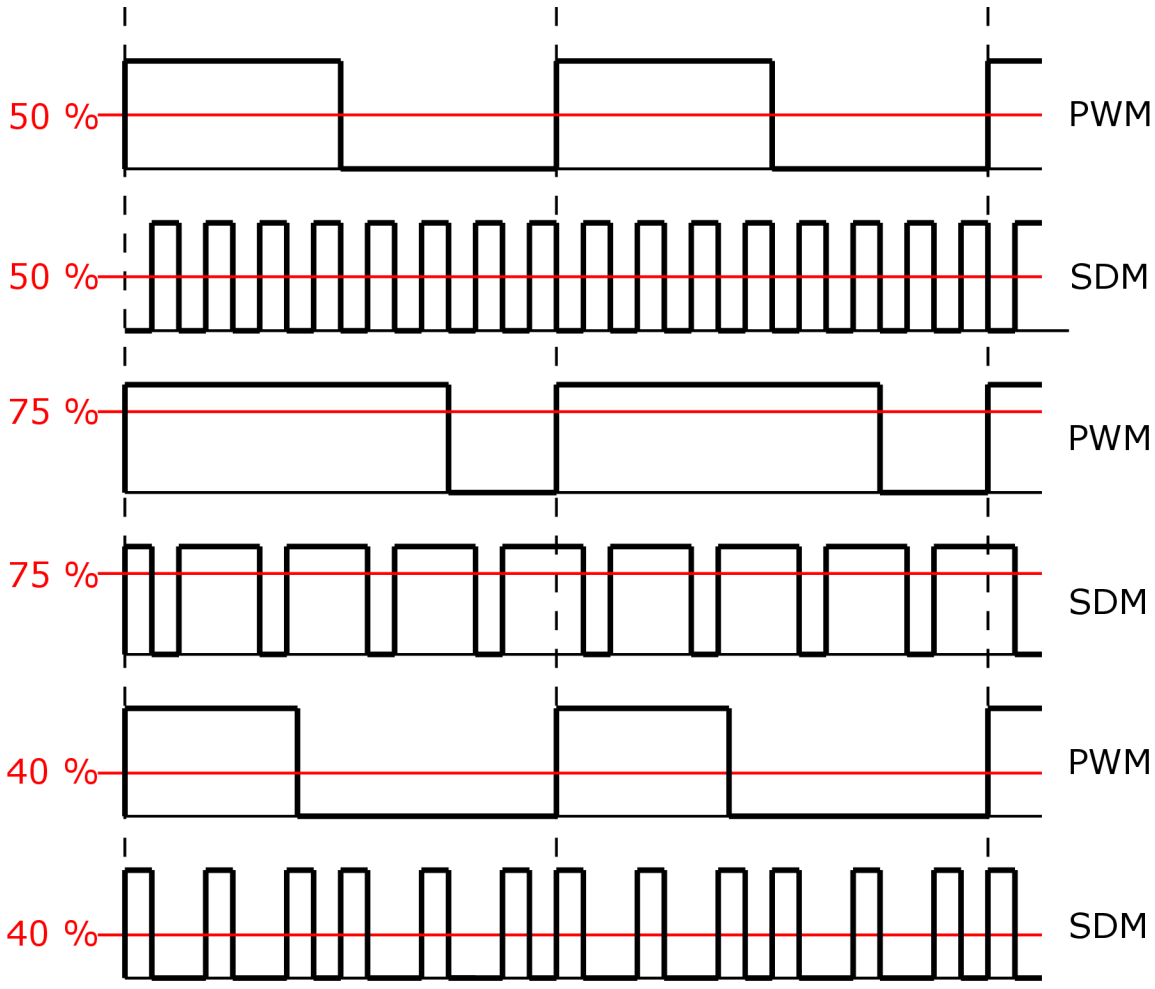
Konečný stavový automat

Konečný stavový automat je obecný model sekvenčního logického obvodu používaný, který se může nacházet v některém ze svých stavů. Na základě aktuálního vnitřního stavu jsou určeny hodnoty výstupních signálů podle výstupní funkce automatu. Následující stav je určený přechodovou funkcí stavového automatu na základě hodnot vstupních signálů. Konečný stavový automat lze využít k realizaci **debounceru**, jehož návrh je obsahem úlohy č. 8.

Sigma-delta modulace

V digitální technice je sigma-delta modulace způsobem pro modulování vstupního signálu obdobně jako pulsně šířková modulace (PWM). Principem pulsně šířkové modulace je změna střídy (poměru doby kdy signál nabývá hodnot log. 1 a 0) výstupního signálu podle hodnoty vstupního signálu. Sigma-delta modulace generuje na výstupu pulsy různé délky, kdy poměr času výstupu v log. 1 a 0 odpovídá hodnotě vstupního signálu. Realizace pulsně šířkového modulátoru je obsahem úlohy č. 7.

Na obrázku A.1 je vidět porovnání výstupního signálu pulsně šířkové modulace a sigma-delta modulace. Poměr doby signálu v log. 1 a 0 je stejný. Výstup sigma-delta převodníku se častěji mění a hodnota na výstupu ze může zdát více odpovídající požadované střední hodnotě.



Obr. A.1: Porovnání výstupu pulsně šířkové a sigma-delta modulace

Vypracování laboratorní úlohy

Úkol č. 1

V návrhovém prostředí Xilinx ISE WebPack navrhněte děličku frekvence umožňující dělení signálu libovolnou hodnotou dělitele. Děličku frekvence realizujte v entitě *clk_divider_tx* v projektu *UART_VHDL*. Dělička bude realizovaná synchronním čítačem *counter_16* pro podělení vstupního signálu *clk_in* hodnotou vstupního vektoru *div_int*. Pro přesné vydělení vstupního signálu desetinnou částí vypočítaného dělitele použijte principu čítače sigma-delta modulace *counter_fr_16*, který bude zapojený v kaskádě s čítačem celé části dělitele. Výslednou frekvenci vydělte 16 pomocí synchronního čítače modulo 16 *counter*. Výstupním signálem bude *terminal_count* tohoto čítače.

```
ENTITY clk_divider_tx IS
GENERIC(
   DIV_INT_BITS: positive := 16; --! Genericky parametr poctu bitu cele casti delitele frekvence
   DIV_FRACT_BITS: positive := 4 --! Genericky parametr poctu bitu desetinne casti delitele frekvence
   );
PORT(
   rst: IN  std_logic;        --! Vstupni signal synchronniho resetu
   clk_in: IN  std_logic;     --! Vstupni hodinovy signal
   div_int: IN std_logic_vector(DIV_INT_BITS - 1 DOWNTO 0); --! Vstupni signal cele casti delitele
   div_fract: IN std_logic_vector(DIV_FRACT_BITS - 1 DOWNTO 0);   --! Vstupni signal desetinne casti delitele
   tc: OUT std_logic          --! Vstupni signal pulsu pozadovane frekvence
   );
END ENTITY clk_divider_tx;
--! @brief Behavioralni popis architektury entity clk_divider_tx
--! @details Entita generuje frekvenci prenosove rychlosti podle nastaveneho delitele. Delicka funguje na principu zlomkove delicky frekvence. Dva citace jsou v kaskade.
--! @details První z nich pocita celou cast delitele frekvence, druhy funguje na principu sigma-delta modulace a inkrementuje se pri kazdem cyklu prvniho citace.
--! @details Frekvence je vydelena 16 poslednim citacem.
ARCHITECTURE Behavioral OF clk_divider_tx IS
   SIGNAL counter_16: unsigned(DIV_INT_BITS - 1 DOWNTO 0) := (OTHERS => '0'); --! Citac cele casti delitele.
   SIGNAL counter_fr_16: unsigned(DIV_FRACT_BITS DOWNTO 0) := (OTHERS => '0');   --! Citac desetinne casti delitele (sigma-delta).
   SIGNAL counter: natural RANGE 0 TO 15 := 0;  --! Citac pro deleni frekvence 16.
   SIGNAL terminal_count: boolean;  --! Signal dopocitani citace counter.
   SIGNAL terminal_count_16: boolean;  --! Signal dopocitani citace counter_16.
BEGIN

END ARCHITECTURE Behavioral;
```

Výsledný návrh upravte i pro děličku frekvence UART přijímače v entitě *clk_divider_rx*. Dělička přijímače bude v chodu pouze pokud bude vstupní signál *run* mít hodnotu log. 1. Dále bude třeba přiřadit výstupu *tc_16* původní frekvenci, která není vydělena čítačem *counter*.

Úkol č. 2

V návrhovém prostředí Xilinx ISE WebPack navrhněte debouncer pro odstranění zákmitů. K návrhu můžete využít řešení úlohy č. 8.

```
ENTITY debouncer IS
PORT(
   input: IN  std_logic;      --! Vstupni signal k odstraneni zakmitu
   clk_in: IN  std_logic;     --! Vstup synchronizacniho hodinoveho signalu
   clk_en: IN  std_logic;     --! Vstup povolovaciho signalu s periodou pro odstraneni zakmitu (10 ms)
   output: OUT std_logic;     --! Vystup bez zakmitu
   ce:     OUT std_logic      --! Vystupni puls s delkou jedne periody synchronizacniho hodinoveho signalu
   );
END ENTITY debouncer;
--! @brief Behavioralni popis entity debouncer
--! @details Realizace debounceru pro odstraneni zakmitu tlacitek pomoci FSM.
ARCHITECTURE Behavioral OF debouncer IS
    TYPE state IS (S0, S1, S2, S3, S4, S5);  --! Enum typ pro stavy FSM
    SIGNAL actual_state: state := S0;  --! Signal s aktualnim stavem FSM
    SIGNAL next_state: state := S0; --! Signal s nasledujicim stavem FSM
BEGIN
END ARCHITECTURE Behavioral;
```

B Zadání studentské laboratorní úlohy pro dokončení a zprovoznění UART

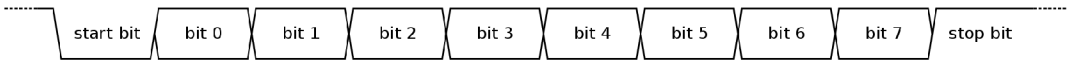
Cíle

- Návrh řídicího stavového automatu přijímače.
- Implementace návrhu do cílového obvodu FPGA a zprovoznění UART.
- Využití UART k příjmu hudebních dat určených pro programovatelný zvukový generátor (PSG).

Teoretický úvod

Sériové komunikační rozhraní UART

UART (Asynchronní sériový vysílač/přijímač) je plně duplexní, dvouvodičová asynchronní sériová komunikační sběrnice. Výhodou asynchronní komunikace je využití pouze dvou datových vodičů **Tx** a **Rx**, popřípadě vodičem **GND** (společná zem). Průběh přenosu datového rámce je na obrázku B.1. Klidový stav sériové linky je v log. 1. Přenos začíná sestupnou hranou start bitu, dále je odesláno zpravidla 8 datových bitů a případně paritní bit. Přenos datového rámce ukončuje stop bit s hodnotou log. 1. Frekvenci přenosu udává hodinový signál. Komponenta pro jeho vygenerování je předmětem předchozí laboratorní úlohy A.



Obr. B.1: Datový rámec UART

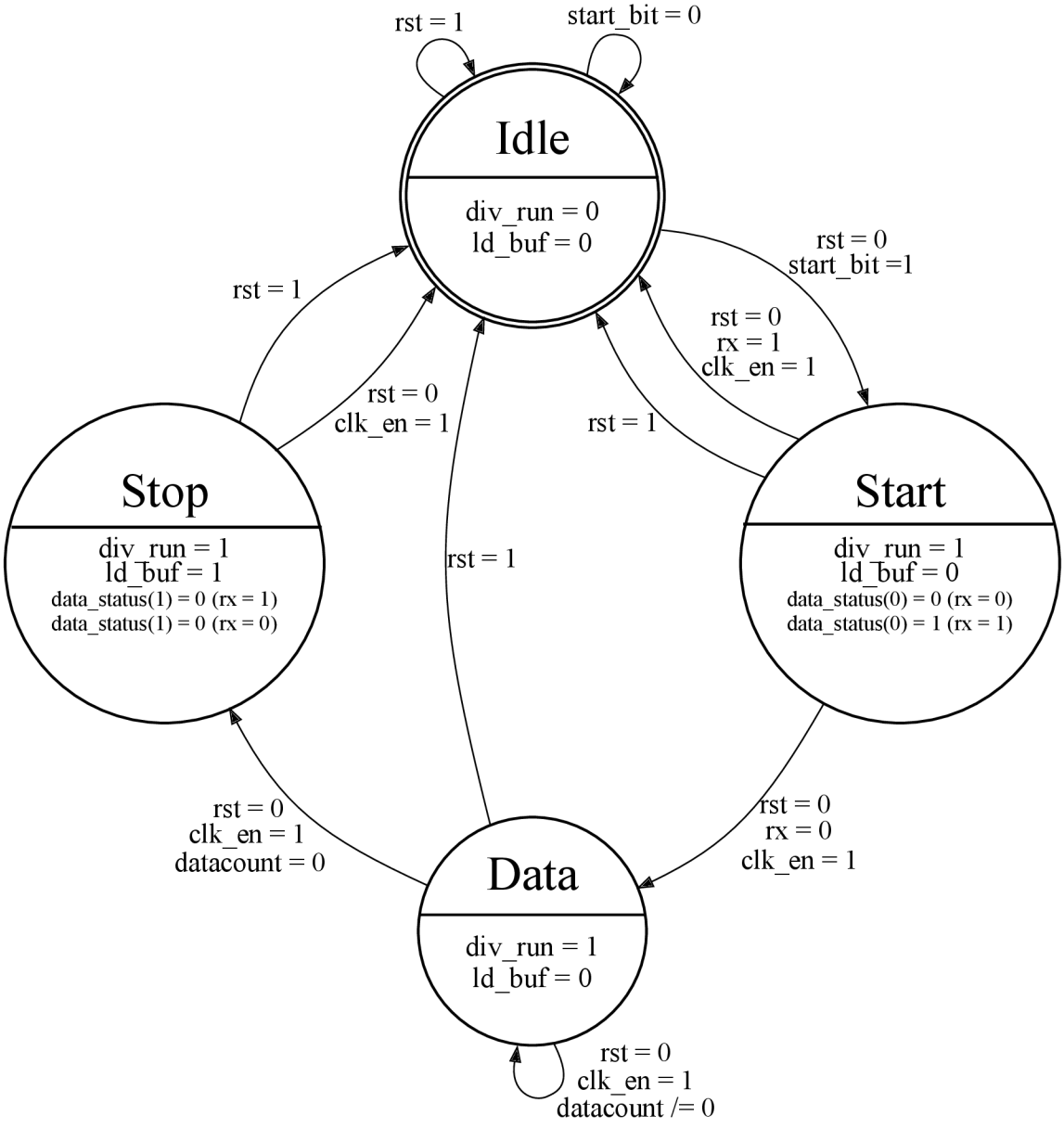
Programovatelný zvukový generátor PSG

Programovatelný zvukový generátor využívá ke generování tónů zápis hodnot do pole registrů, které nastavují výstupní signály jednotlivých kanálů. Návrh PSG je předmětem laboratorní úlohy zabývající jeho zprovozněním.[3]

Vypracování laboratorní úlohy

Úkol č. 1

V návrhovém prostředí Xilinx ISE WebPack navrhněte řídicí stavový automat přijímače UART. Návrh realizujte v entitě *receiver_fsm* v projektu *UART_VHDL*. Stavový automat navrhněte podle stavového diagramu na obrázku B.2.



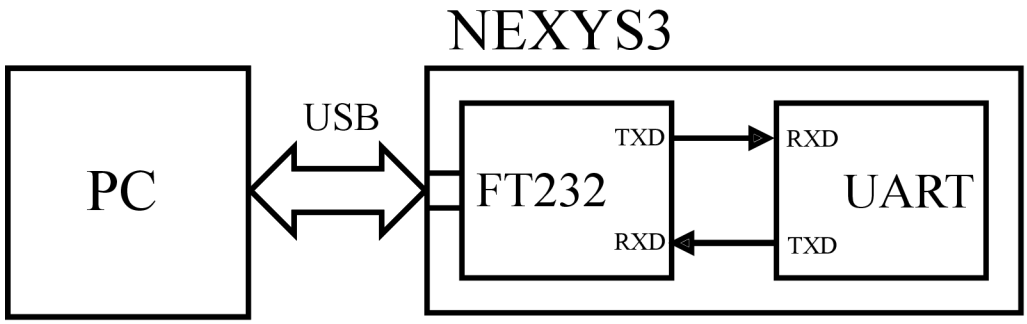
Obr. B.2: Stavový diagram řídicího stavového automatu přijímače

```
ENTITY receiver_fsm IS
PORT(
    clk_in: IN std_logic;  --! Vstupni hodinovy signal
    rx: IN std_logic;   --! Vstupni signal s filtrovanym signalem seriovych dat
    start_bit: IN std_logic;  --! Vstupni signal detekujici start bit
    rst: IN std_logic;  --! Vstupni signal synchronniho resetu
    clk_en: IN std_logic;  --! Vstupni povolovaci signal frekvence prenosove rychlosti
    use_par: IN std_logic; --! Vstupni signal volby patity
    div_run: OUT std_logic;   --! Vystupni signal pro chod delicky frekvence
    ld_buf: OUT std_logic;    --! Vystupni signal pro presun prijatych dat do bufferu
    data_status: OUT std_logic_vector (1 DOWNTO 0) := (OTHERS => '0') --! Vystupni registr se stavem start bitu a stop bitu
    );
END ENTITY receiver_fsm;
--! @brief Behavioralni popis architektury entity receiver_fsm
--! @details Entita realizuje ridici stavovy automat prijimace. Architektura entity je resena tremi procesy stavoveho automatu.
ARCHITECTURE Behavioral OF receiver_fsm IS
    TYPE state IS (idle, start, data, stop); --! Vyctovy typ pro stavy fsm
    SIGNAL actual_state: state := idle;   --! Interni signal aktualniho stavu fsm
    SIGNAL next_state: state := idle;    --! Interni signal nasledujiciho stavu fsm
    SIGNAL datacount: unsigned(3 DOWNTO 0) := "0111"; --! Interni signal citace odesilanych bitu
BEGIN

END ARCHITECTURE Behavioral;
```

Úkol č. 2

Po realizaci všech předchozích úkolů je projekt *UART_VHDL* kompletní. Dle blokového schématu na obrázku B.3 připojte vývojovou desku k PC. Otestujte funkčnost návrhu pomocí terminálu Putty v PC.



Obr. B.3: Blokové schéma laboratorní úlohy realizující spojení navrženého vysílače/přijímače s PC

Úkol č. 3

Navržené entity z předchozích úloh využijte v dodaném projektu *UART_PSG*. Vyzkoušejte přehrávání hudebních dat pomocí programu v PC.

C Obsah elektronické přílohy

- `/` ..... kořenový adresář elektronické přílohy
  - `UART_VHDL` ..... adresář s projektem UART_VHDL
    - `anode_refresher.vhd` entita přepínání anod 7-seg. displeje (převzatá z kurzu BPC-LOS)
    - `clk_divider.vhd` ..... entita děličky frekvence pro debouncery tlačítek
    - `clk_divider_tx.vhd` ..... entita děličky frekvence vysílače
    - `clk_divider_rx.vhd` ..... entita děličky frekvence přijímače
    - `debouncer.vhd` ..... entita debounceru pro odstranění zákmitů tlačítek
    - `filter.vhd` ..... entita filtru pro odstranění metastability přijímače
    - `four_digit_display.vhd` ..... entita pro zobrazování přijatých dat na 7-seg. displeji
    - `parity.vhd` ..... knihovní balík s funkcemi generování a kontroly parity
    - `receiver_fsm.vhd` ..... entita se stavovým automatem přijímače
    - `shiftregister_rx.vhd` ..... entita posuvného registru přijímače
    - `shiftregister_tx.vhd` ..... entita posuvného registru vysílače
    - `start_detector.vhd` ..... entita start detektoru přijímače
    - `testbench_div_rx.vhd` ..... testbench děličky frekvence přijímače
    - `testbench_div_tx.vhd` ..... testbench děličky frekvence vysílače
    - `testbench_filter.vhd` ..... testbench filtru přijímače
    - `testbench_rx_fsm.vhd` ..... testbench stavového automatu přijímače
    - `testbench_sr_rx.vhd` ..... testbench posuvného registru přijímače
    - `testbench_sr_tx.vhd` ..... testbench posuvného registru vysílače
    - `testbench_start.vhd` ..... testbench start detektoru přijímače
    - `testbench_tx_fsm.vhd` ..... testbench stavového automatu vysílače
    - `transmitter_fsm.vhd` ..... entita stavového automatu vysílače
    - `UART.vhd` ..... strukturální popis top entity UART
    - `uart_cd.cdc` ..... konfigurační soubor ChipScope Pro
    - `uart_constraints.ucf` soubor s konfigurací připojení periferií vývojové desky k návrhu
    - `uart_rx.vhd` ..... strukturální popis přijímače
    - `uart_tx.vhd` ..... strukturální popis vysílače
    - `wave_rx_div.wcfg` konfigurace průběhu simulace děličky frekvence přijímače
    - `wave_rx_flt.wcfg` ..... konfigurace průběhu simulace filtru přijímače
    - `wave_rx_fsm.wcfg` ..... konfigurace průběhu simulace stavového automatu přijímače pro možnost vygenerování průběhu na obrázku 3.19
    - `wave_rx_fsm2.wcfg` ..... konfigurace průběhu simulace stavového automatu přijímače pro možnost vygenerování průběhu na obrázku 3.20
    - `wave_rx_sr1.wcfg` konfigurace průběhu simulace posuvného registru přijímače pro možnost vygenerování průběhu na obrázku 3.14
    - `wave_rx_sr1_det.wcfg` ..... konfigurace průběhu simulace posuvného registru přijímače pro možnost vygenerování průběhu na obrázku 3.15
    - `wave_rx_sr2.wcfg` konfigurace průběhu simulace posuvného registru přijímače pro možnost vygenerování průběhu na obrázku 3.16

- `/` ..... kořenový adresář elektronické přílohy
  - `UART_VHDL` ..... adresář s projektem UART_VHDL
    - `wave_rx_sr3.wcfg` konfigurace průběhu simulace posuvného registru přijímače pro možnost vygenerování průběhu na obrázku 3.17
    - `wave_rx_start.wcfg` ..... konfigurace průběhu simulace detektoru start bitu přijímače
    - `wave_tx_div.wcfg` ... konfigurace průběhu simulace děličky frekvence vysílače
    - `wave_rx_fsm.wcfg` konfigurace průběhu simulace stavového automatu vysílače pro možnost vygenerování průběhu na obrázku 3.6
    - `wave_rx_fsm2.wcfg` ..... konfigurace průběhu simulace stavového automatu vysílače pro možnost vygenerování průběhu na obrázku 3.7
    - `wave_rx_sr1.wcfg` . konfigurace průběhu simulace posuvného registru vysílače pro možnost vygenerování průběhu na obrázku 3.3
    - `wave_rx_sr2.wcfg` . konfigurace průběhu simulace posuvného registru vysílače pro možnost vygenerování průběhu na obrázku 3.4
  - `Dokumentace_UART_VHDL.pdf` .. dokumentace projektu UART_VHDL – Doxygen
  - `Baudrate.xlsx` ..... excel tabulka pro výpočet nastavení přenosové rychlosti
  - `UART_PSG` ..... adresář s projektem UART_PSG pro přehávání hudbních dat
    - `data_control.vhd` ..... entita pro seskupení dat registrů PSG
    - `reg_write_fsm.vhd` ..... entita pro zápis dat do registrů PSG
    - `custom_types.vhd` ..... knihovní balík s typem pole registrů pro PSG
    - ..... ostatní zdrojové soubory realizují UART a PSG (nejsou tu již uváděny)
  - `Videa` ..... adresář s demonstračními videi
    - `256kbaud_putty.mp4` ..... video demonstrující přenos dat mezi PC – UART
    - `Bervrlycp.mp4` ..... nahrávka delší části 1. skladby
    - `Ghostbusters.mp4` ..... nahrávka delší části 2. skladby
    - `Starwars.mp4` ..... nahrávka delší části 3. skladby
  - `thesis.pdf` ..... elektronická verze bakalářské práce